PROGRAMACION CON FORTRAN

Incluye Fortran Estructurado

TEORIA y
375 problemas
resueltos

SEYMOUR LIPSCHUTZ

ARTHUR POE

Prólogo

La segunda mitad del siglo veinte bien puede calificarse como la era de los computadores. El hombre, en casi todos los aspectos del conocimiento —tales como la ingeniería, las ciencias físicas, la economía, la psicología, la educación, las ciencias sociales, las ciencias médicas, las leyes y los negocios, es decir, donde quiera que se necesite recolectar datos y analizarlos por medio de cálculos— tendrá algún contacto con los computadores y los lenguajes que se utilizan para darles instrucciones. Uno de tales lenguajes de programación se conoce con el nombre de Fortran, que es la sigla de las palabras FORmula TRANslation. Este libro ha sido diseñado y escrito para hacer una introducción al lenguaje Fortran y para enseñar la solución de problemas a partir de él. Además de presentar la sintaxis del Fortran, el objetivo principal del texto es enseñar al lector cómo se escriben clara y eficientemente los programas Fortran, con la ayuda de las técnicas, así como también de las prácticas de la programación. En este texto se desarrollan todos los principios del Fortran propiamente dicho, lo mismo que los aspectos importantes del Fortran estructurado. Este libro es recomendable como texto para un curso de introducción a la programación Fortran o como complemento de los textos normales de un curso de "Introducción a Ciencias de la Computación".

La deliberada dosificación del tema así como la presentación de muchos ejemplos escalonados, permitirán que este texto despierte el interés de muchas personas y sirva particularmente como una guía efectiva del aprendizaje autodidacta. Cada capítulo comienza con una clara exposición de las definiciones y de los principios pertinentes, junto con el material ilustrativo y descriptivo. Esto se complementa con grupos escalonados de problemas resueltos y de problemas suplementarios. Los problemas resueltos ilustran y amplían el tema, mientras que los problemas suplementarios suministran un completo repaso del material del capítulo.

El texto se divide en doce capítulos. En el capítulo 1 se presenta una breve discusión de las operaciones básicas del computador, por medio de ejemplos sencillos. Esto le da al lector una visión del lenguaje Fortran, así como también alguna experiencia sobre la dinámica que encierra la programación Fortran. En este capítulo también se discute la forma como se perforan las instrucciones y cómo se arregla el paquete Fortran, de tal manera que el lector pueda escribir y manejar programas sencillos lo más pronto que le sea posible. La introducción formal al lenguaje Fortran comienza en el capítulo 2, donde se muestran las expresiones aritméticas y las proposiciones de asignación aritmética. Se discute tanto la aritmética de enteros como la de reales, así como también los inconvenientes de la aritmética del computador. El capítulo 3 hace referencia a las operaciones de entrada/salida numérica. Se discute la entrada/salida con formato y sin formato.

En el capítulo 4 se introduce la diagramación para ayudar al lector a resaltar y visualizar los procedimientos de solución de los problemas. El capítulo 4 discute las proposiciones de transferencia de control en el Fortran básico (no estructurado), incluyendo la proposición IF lógica. En el capítulo 12 se discuten aspectos relacionados con el control estructurado y con la proposición IF en bloque. Análogamente, en el capítulo 5 se muestra el ciclo DO básico con índice, mientras que en el capítulo 12 se tratan el ciclo DO generalizado y las estructuras WHILE y FOR. En el capítulo 6 se discuten los arreglos y las variables subindicadas. Para ayudar al lector a escribir programas complejos en módulos, en el capítulo 7 se discuten los subprogramas FUNCTION y SUBROUTINE.

En el capítulo 8 se tratan las técnicas de programación para búsqueda, clasificación, mantenimiento de archivos, etc. También se discuten cálculos numéricos, tales como determinación de los ceros de funciones y soluciones de sistemas de ecuaciones lineales. En el capítulo 9 se trata la información en forma de caracteres y las variables lógicas, junto con su forma de entrada/salida. En el capítulo 10 se cubren aspectos adicionales de entrada/salida, tales como el campo G y el formato en tiempo de ejecución. En el capítulo 11 se presentan otros aspectos del Fortran, tales como las variables de DOBLE PRECISION, las variables COMPLEJAS y las proposiciones COMMON y EQUIVALENCE. Como se dijo anteriormente, en el capítulo 12 se discuten aspectos del Fortran estructurado, tales como la proposición IF en bloque y el ciclo DO generalizado. Este

capítulo es compatible con los recientes cambios del Fortran normal y con los compiladores WATFOR y WATFIV.

Los temas y los capítulos se han escrito de tal manera que ellos sean independientes, tanto como sea posible, de otros temas y capítulos. Esto se ha hecho con el ánimo de que el libro sea más flexible y con el deseo de proporcionar un texto más útil como referencia. También hemos incluido un apéndice sobre la representación interna de los datos en el computador.

Deseamos agradecer a nuestros muchos amigos y colegas, sus invaluables sugerencias y la revisión crítica del manuscrito. También deseamos expresar nuestra gratitud al personal de McGraw-Hill Schaum's Outline Series, en particular a John Aliano y Ellen LaBarbera, por su inagotable colaboración.

Seymour Lipschutz
Arthur T. Poe

Contenido

Capítulo 1 **INTRODUCCION, ORGANIZACION DE UN PROGRAMA** 1

1.1 Introducción 1.2 Perforación de las proposiciones Fortran 1.3 El paquete Fortran 1.4 Almacenamiento de números 1.5 Lectura de datos 1.6 Ejecución de decisiones 1.7 Reseña de los computadores y de los lenguajes.

Capítulo 2 **PROPOSICIONES ARITMETICAS** 19

2.1 Introducción 2.2 Constantes numéricas (números) 2.3 Nombres de variables (nombres de las localizaciones de almacenamaiento) 2.4 Proposiciones de declaración de tipo—entera, real 2.5 Operaciones aritméticas—aritmética de entero y de reales 2.6 Expresiones aritméticas 2.7 Operaciones con mezcla de modos 2.8 Funciones matemáticas internas 2.9 Proposición de asignación aritmética 2.10 Aritmética del computador.

Capítulo 3 **ENTRADA/SALIDA NUMERICA** 41

3.1 Introducción 3.2 Entrada/salida sin formato 3.3 Introducción a la entrada/salida con formato 3.4 Especificaciones de campo para entrada y entrada con formato 3.5 Proposiciones WRITE con formato y control de carro 3.6 Especificación de campos de salida 3.7 Campo literal 3.8 Registros, registros múltiples, diagonal 3.9 Factor de repetición 3.10 Un programa completo y sencillo.

Capítulo 4 **TRANSFERENCIA DE CONTROL, DIAGRAMAS DE FLUJO** 77

4.1 Introducción 4.2 Transferencia incondicional 4.3 Transferencia condicional 4.4 Expresiones de relación 4.5 Proposición IF lógica 4.6 Control de un ciclo 4.7 Proposición IF aritmética 4.8 Proposición GO TO computada 4.9 Algoritmos 4.10 Tarjetas de encabezamiento y de delimitación de datos.

Capítulo 5 **CICLOS DO** 113

5.1 Introducción 5.2 Proposición CONTINUE 5.3 Usos sencillos de la proposición DO 5.4 Proposición DO 5.5 Reglas acerca del uso del ciclo DO 5.6 Salida de un ciclo DO 5.7 Transferencia dentro de y hacia un ciclo DO 5.8 Necesidad de la proposición CONTINUE 5.9 Ciclos DO anidados.

Capítulo 6 **ARREGLOS, VARIABLES SUBINDICADAS** 140

6.1 Introducción 6.2 Arreglos unidimensionales 6.3 Proposiciones DIMENSION 6.4 Expresiones aritméticas para subíndices 6.5 Ejemplos de utilización de arreglos 6.6 Arreglos multidimensionales 6.7 Entrada/salida de un arreglo, ciclos DO implícitos 6.8 Ejemplo de la técnica de programación.

Capítulo 7 **FUNCIONES Y SUBRUTINAS** 172
7.1 Introducción 7.2 Subprogramas FUNCTION 7.3 Llamado de subprogramas FUNCTION 7.4 Subprogramas FUNCTION para el cálculo de varios valores 7.5 Arreglos y subprogramas FUNCTION, dimensión variable 7.6 Proposición que define funciones aritméticas 7.7 Subrutinas 7.8 SUBRUTINA contra FUNCTION.

Capítulo 8 **TECNICAS DE PROGRAMACION Y CALCULO NUMERICO** 194
8.1 Introducción 8.2 Clasificación 8.3 Intercalación 8.4 Búsqueda 8.5 Actualización 8.6 Método de Horner 8.7 Integración numérica 8.9 Vectores y matrices 8.10 Ecuaciones lineales.

Capítulo 9 **INFORMACION EN FORMA DE CARACTERES, VARIABLES LOGICAS Y OPERACIONES** 217
9.1 Introducción 9.2 Almacenamiento de caracteres 9.3 Campo A 9.4 Manejo de la información en forma de caracteres 9.5 Campo H 9.6 Constantes lógicas y variables lógicas 9.8 Asignación de valores lógicos 9.9 Campo L 9.10 Jerarquía de operaciones.

Capítulo 10 **ASPECTOS ADICIONALES DE ENTRADA/SALIDA** 239
10.1 Introducción 10.2 Proposición DATA 10.3 Campo T 10.4 Campo G 10.5 Factor de escala 10.6 Regla del paréntesis izquierdo 10.7 Formatos en tiempos de ejecución 10.8 Elaboración de gráficos.

Capítulo 11 **ASPECTOS DIVERSOS DEL FORTRAN** 264
11.1 Introducción 11.2 Proposiciones de declaración de tipo 11.3 Proposición IMPLICIT 11.4 Doble precisión 11.5 Números complejos 11.6 Proposición GO TO asignada. Proposición ASSIGN 11.7 Entradas múltiples y retornos a un subprograma 11.8 Proposiciones COMMON no rotuladas 11.9 Proposiciones COMMON rotuladas 11.10 Proposiciones EQUIVALENCE 11.11 BLOCK DATA 11.12 EXTERNAL.

Capítulo 12 **FORTRAN ESTRUCTURADO** 288
12.1 Estructuras IF 12.2 Estructuras de control de ciclos.

Apéndice A **FUNCIONES DE BIBLIOTECA** 303

Apéndice B **REPRESENTACION INTERNA DE LOS DATOS** 305

INDICE 311

Capítulo 1

Introducción, organización de un programa

1.1 INTRODUCCION

La segunda mitad del siglo veinte bien puede calificarse como la era de los computadores. Estos aparatos y los diversos "lenguajes" utilizados para darles instrucciones, se han popularizado tanto como las máquinas de escribir y las reglas de cálculo. Esencialmente, un computador desarrolla tres funciones:

1. Recibe datos (entrada).
2. Procesa datos por medio de variados cálculos.
3. Emite datos (salida).

Estas funciones se llevan a cabo por medio de un *programa* que se da al computador y que consiste en una secuencia de *proposiciones*. (Una secuencia parcial de proposiciones recibe el nombre de *segmento de programa*.)

El propósito principal de este texto es enseñar al lector a escribir programas de computador, utilizando el lenguaje FORTRAN. (La palabra FORTRAN proviene de FORmula TRANslation.) Este estudio se iniciará formalmente en el próximo capítulo; el presente cubrirá ciertos aspectos de procedimiento relativos a la perforación de las proposiciones Fortran sobre tarjetas y a la forma como se ordena el paquete Fortran para el computador. Aprenderemos también algunos aspectos de los computadores que pueden darnos un poco de experiencia acerca de la dinámica que encierra la programación Fortran. En particular, veremos cómo ejecuta un computador algunos programas sencillos. (En efecto, recalcamos que el lector debería perforar uno o más de estos programas y correrlos en un computador.)

Concluimos el capítulo con una reseña de los computadores y de los lenguajes de éstos. Lo anterior implica muchos detalles que no se requieren en absoluto para algunos de los tópicos siguientes de este libro. De ahí que el lector puede tocar ligeramente esta sección con una primera lectura y utilizarla como referencia cuando lo crea necesario.

1.2 PERFORACION DE LAS PROPOSICIONES FORTRAN

Algunas veces los programas Fortran se escriben inicialmente en hojas de codificación. (Véase la figura 1-1.) Sin embargo, una vez escritos, deben suministrarse al computador por medio de alguna forma legible para una máquina, tal como un paquete de tarjetas perforadas, una cinta de papel, una cinta magnética o por medio de un terminal con máquina de escribir, dependiendo cada caso del dispositivo de entrada que se utilice. A través de este libro, supondremos que nuestro dispositivo de entrada es una lectora de tarjetas, por lo que los programas estarán perforados en ellas. Una tarjeta, como la que se muestra en la figura 1-2(*a*), tiene 80 columnas, en cada una de las cuales puede colocarse un caracter sencillo.

Cada proposición de un programa Fortran debe perforarse en tarjetas separadas. Como se ilustra en la figura 1-2(*b*), no todas las 80 columnas de la tarjeta se utilizan para la proposición Fortran. Específicamente, tenemos las siguientes reglas:

1. La proposición Fortran puede perforar solamente entre las columnas 7 y 72, inclusive.
2. La columna 6 se denomina columna de continuación y está reservada para el siguiente propósito. Si una proposición Fortran es tan extensa que no cabe en las 66 columnas mencionadas anteriormente, entonces puede continuarse (entre las columnas 7 y 72) hasta completar 19 tarjetas adicionales. En tal caso, debe perforarse un "1" (o cualquier símbolo diferente de cero), en la columna 6 de cada una de las tarjetas adicionales. Esto es, cualquier símbolo (diferente de cero)

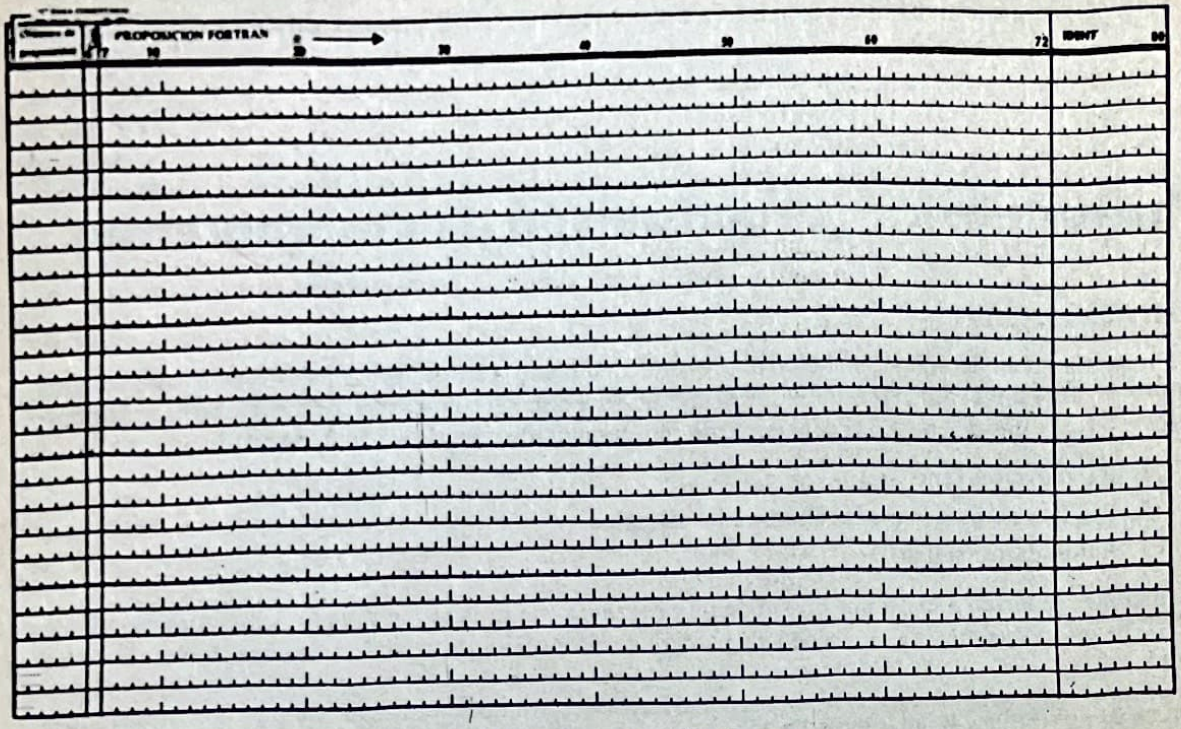
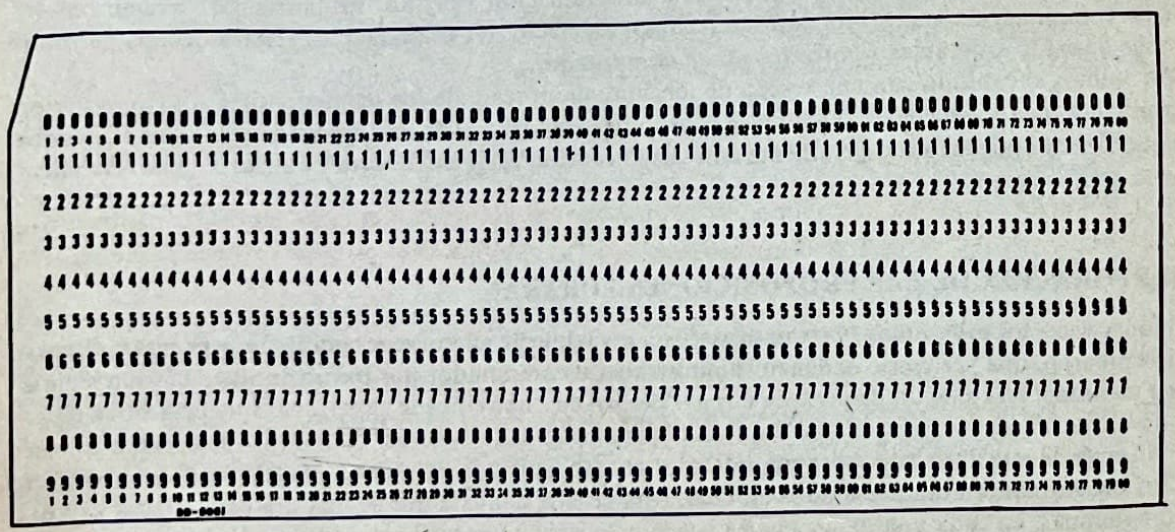
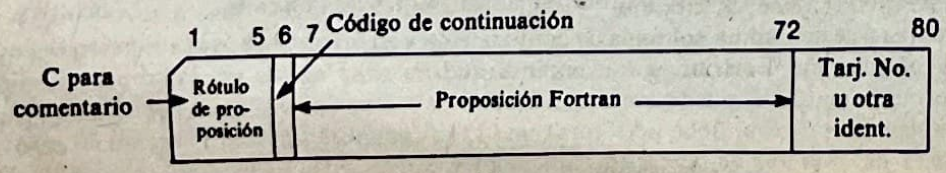


Fig. 1-1 Hoja de codificación Fortran.



(*a*)



(*b*)

Fig. 1-2

perforado en la columna 6, le informa al computador que la proposición es continuación de la última proposición.

3. Las proposiciones pueden rotularse para hacer referencia a ellas y en este caso los rótulos se perforan entre las columnas 1 y 5 de la proposición rotulada. En Fortran, un rótulo es cualquier entero positivo sin signo (con un máximo de 5 dígitos), que recibe el nombre de *número de proposición.* Claramente, dos proposiciones diferentes no pueden rotularse con el mismo número de proposición.
4. El computador ignora el contenido de las columnas 73 a 80. Estas columnas pueden perforarse para identificación del programa, secuencia de las instrucciones o para cualquier otro propósito.

Los *comentarios* pueden adicionarse a un programa, perforando la letra "C" en la columna 1 de la tarjeta. El comentario, en sí, se puede perforar entre las columnas 2 y 80. El programa no tiene en cuenta los comentarios; por lo común, y usualmente, se utilizan para identificar y explicar un programa. Sencillamente el computador salta un comentario y va a la siguiente proposición. Sin embargo, el comentario aparecerá en la hoja impresa cuando el programa sea imprimido por el computador.

Nota. El lector notará, cuando se perforan las tarjetas, que solamente las letras mayúsculas están disponibles para los caracteres alfabéticos. Los símbolos de la suma, resta, multiplicación y división se escriben como +,—, *, /, respectivamente. La exponenciación, que eleva un número a una potencia, se escribe como **. (Estas operaciones se discuten con más detalle en el capítulo 2.) Así, en Fortran,

$a \cdot b$ se escribe como A*B, $\frac{a}{b}$ como A/B, y a^b como A**B

Observemos que el computador ignora los espacios en blanco que aparecen en las proposiciones Fortran. Por ejemplo, podríamos perforar

A**2 + B**2 en lugar del equivalente A**2+B**2

4*I + 6*J en lugar del equivalente 4*I+6*J

y GO TO en vez de GOTO.

1.3 EL PAQUETE FORTRAN

El programa Fortran que escribimos y perforamos en las tarjetas es sólo parte del paquete que suministramos al computador cuando el programa va a ser ejecutado. Este paquete consiste en la siguiente secuencia de tarjetas:

1. Una tarjeta ID. Esta tarjeta usualmente contiene el nombre del programador, proyecto, curso o número de la cuenta, número de prioridad, máximo tiempo de ejecución, etc.
2. El paquete Fortran. Este paquete de tarjetas contiene el programa Fortran.
3. El paquete de datos. Este paquete de datos, que puede no tenerlos, contiene los datos que lee el computador.
4. La tarjeta de final de archivo. Esta tarjeta indica el fin del paquete.

Los cuatro grupos de tarjetas mencionados anteriormente, pueden estar separados por una o más tarjetas de control, como se ilustra en la figura 1-3. La información exacta sobre la tarjeta ID, tarjeta de final de archivo, y tarjetas de control, dependerá en particular de las facilidades de computación a que tengamos acceso.

El paquete de datos contiene los datos de entrada. En la perforación de una tarjeta de datos, podemos utilizar todas las 80 columnas. Esto es, las reglas sobre perforación de las proposiciones Fortran no se aplican en este caso, y ninguna de las columnas, por ejemplo la 6, tiene propósito especial.

1.4 ALMACENAMIENTO DE NUMEROS

Para el almacenamiento de datos, todo computador tiene una unidad de memoria que consta de muchas *celdillas (localizaciones) de memoria.* Un número puede asignarse a una localización de memoria, solamente

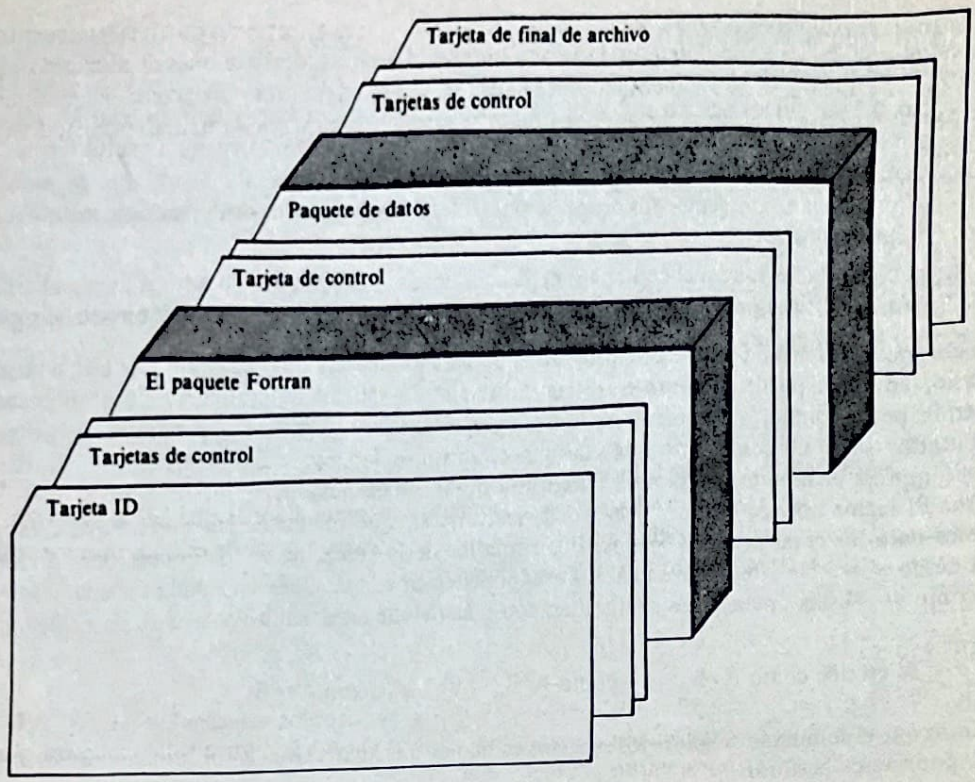


Fig. 1-3 El paquete Fortran

si esa localización recibe un nombre. Representamos la asignación de un número, digamos 75, a una localización de memoria llamada N, por

N ← 75

Más generalmente,

NAME ← expresión aritmética

indica que el valor de la expresión aritmética se asigna a la localización de memoria cuyo nombre es NAME.

Recalcamos que la flecha ← se acepta universalmente como "asignación". Sin embargo, los diferentes lenguajes de programación representan ← de distintas maneras. En Fortran, ← se representa por

=

(es decir el signo igual). Así, la proposición Fortran

I = 7

le indica al computador que debe asignar el número 7 a la localización de memoria llamada I, y la proposición Fortran

J = 4 + 5 + 8

le indica al computador que:

1. Sume los valores a la derecha de =.
2. Almacene el resultado, cuyo valor es 17, en una localización llamada J.

Observe que aquí se incluyen dos acciones: la evaluación de la expresión a la derecha de = y el almacenamiento del resultado en la localización que se da a la izquierda de =.

Una expresión aritmética también puede incluir los nombres de localizaciones de almacenamiento. Sin embargo, estos nombres deben *definirse*, es decir, son los nombres de las localizaciones de almacenamiento, en las cuales los números se han almacenado previamente, dentro del mismo programa. El computador evalúa la expresión aritmética, reemplazando los nombres por los valores que se han almacenado en estas localizaciones.

Por ejemplo, consideremos la proposición Fortran

```
K = 3*I + 2*J - 24
```

donde I y J contienen los valores 7 y 17 respectivamente. (Recuerde que en Fortran el asterisco * significa multiplicación.) El computador evalúa la expresión aritmética a la derecha de = como

$$3 \cdot 7 + 2 \cdot 17 - 24$$

El resultado, que es 31, se almacena en la localización de memoria llamda K. Note que los contenidos de I y de J no se obtienen literalmente de las localizaciones I y J cuando el computador evalúa 3*I + 2*J − 24, sino que los contenidos se copian en la unidad aritmética, e I y J contienen, a pesar de eso, los mismos valores.

Ilustramos estas ideas en nuestro primer programa.

Primer programa

Escribimos un programa que calcula la suma LS y el producto LP de los números I = 24, J = 3, K = 7. La figura 1-4(*a*) es la hoja de codificación que contiene estos cálculos y la figura 1-4(*b*) muestra cómo se miraría el programa, si se perforara en un paquete de tarjetas.

Observe que cada proposición se escribe en una línea separada en la hoja de codificación y se perfora en una tarjeta separada. Seguiremos al computador para observar cómo lee y ejecuta el programa. Sin embargo, primero, aquí se hace una breve explicación de algunas proposiciones del programa:

1. La primera tarjeta tiene la letra "C" perforada en la columna 1, lo que nos indica que se trata de una tarjeta de comentario. Como anotamos anteriormente, el programa no tiene en cuenta los comentarios. Cuando el computador encuentra la letra "C" en la columna 1, simplemente imprime el comentario y prosigue con la siguiente instrucción.
2. La proposición PRINT

   ```
   PRINT, I, J, K, LS, LP
   ```

   le informa al computador que debe imprimir los contenidos de las localizaciones de memoria, I, J, K, LS y LP, en el orden en que aparecen. Esta es una proposición de salida sin formato; los valores se imprimen en algún formato predefinido. Las proposiciones de formato y de formato libre se estudian con detalle en el capítulo 3.
3. La proposición STOP le dice al computador que termine la ejecución; la proposición END siempre es la última proposición de cualquier programa. Véase la sección 1.8(*c*) para una discusión más amplia de sus diferencias.

Ejecución del programa

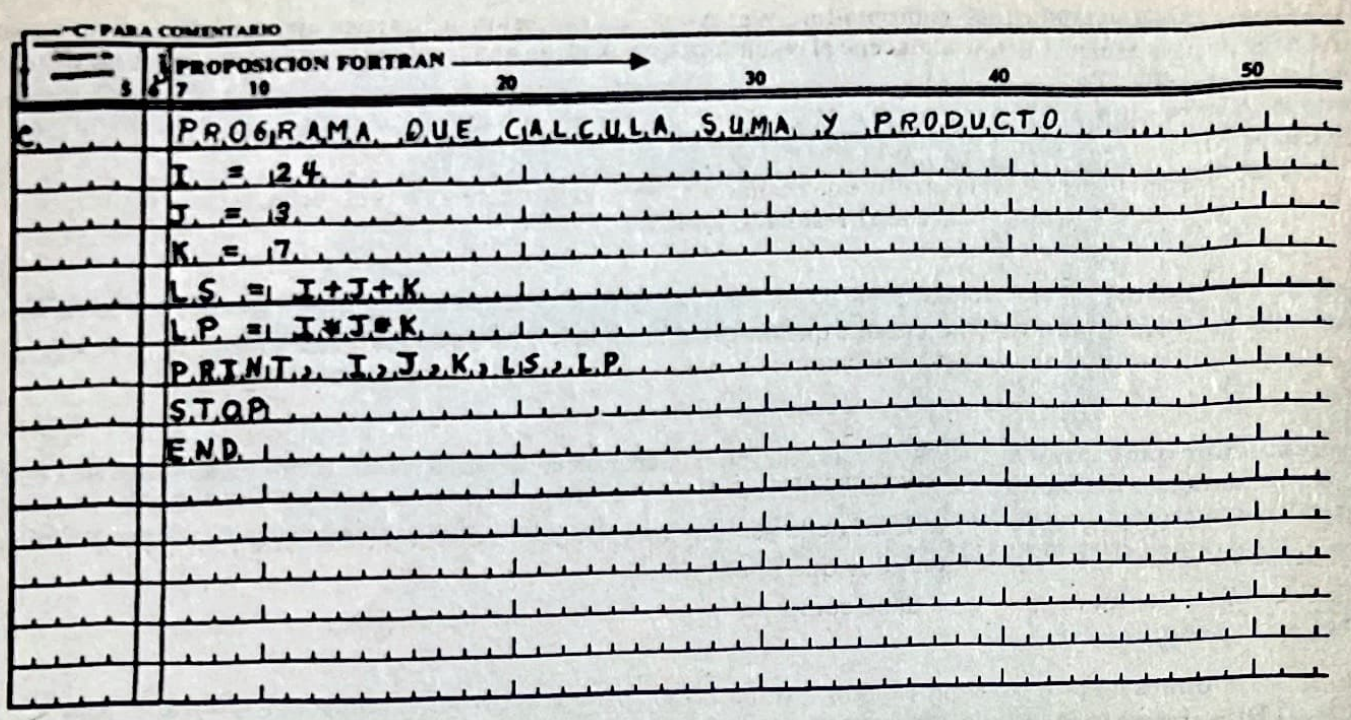
Línea 1. Esta es un comentario, puesto que hay una C escrita en la primera columna. Como se estableció antes, el computador salta esta proposición y va a la siguiente.

Línea 2. Esta le dice al computador que almacene el entero 24 en una localización llamada I:

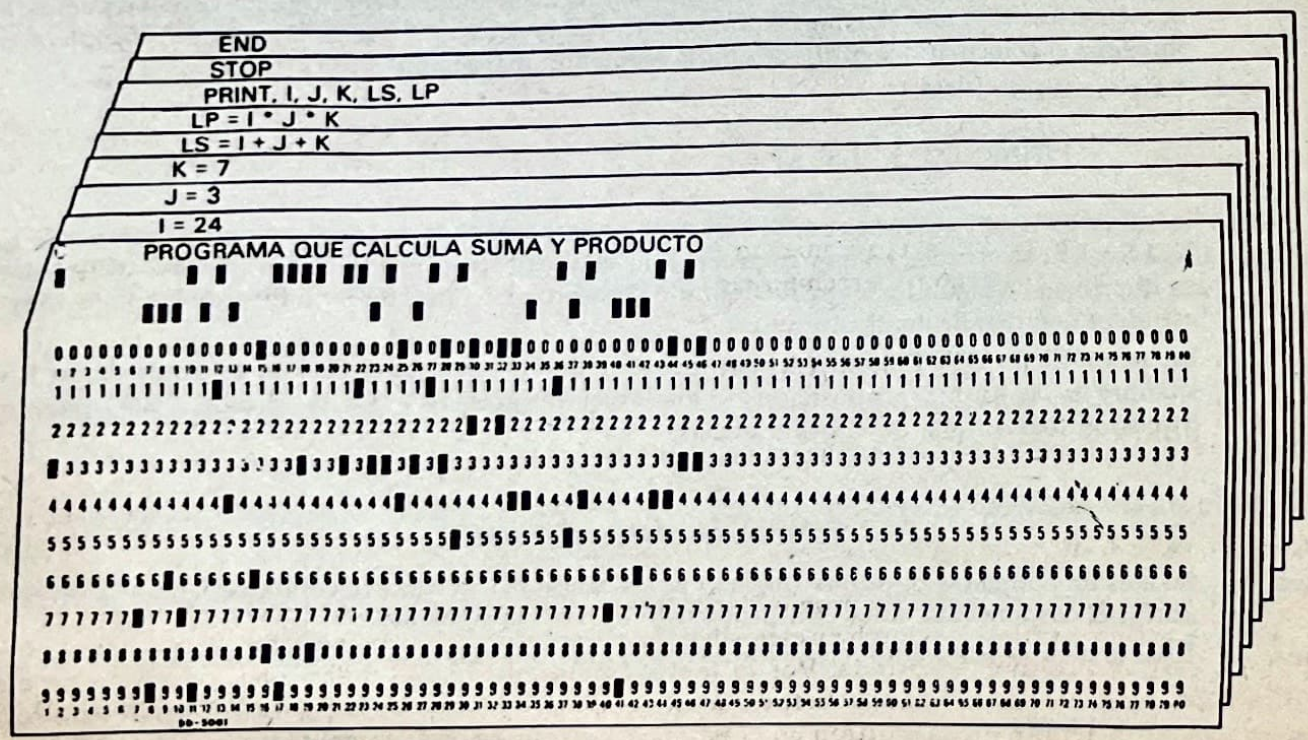
```
I ←24
```

Línea 3. `J ←3`

Línea 4. `K← 7`


"C" PARA COMENTARIO
PROPOSICION FORTRAN
5 6 7 10 20 30 40 50
C PROGRAMA QUE CALCULA SUMA Y PRODUCTO
I = 24
J = 3
K = 7
LS = I+J+K
LP = I*J*K
PRINT, I, J, K, LS, LP
STOP
END

(*a*)


END
STOP
PRINT, I, J, K, LS, LP
LP = I * J * K
LS = I + J + K
K = 7
J = 3
I = 24
PROGRAMA QUE CALCULA SUMA Y PRODUCTO

(*b*)

Fig. 1-4

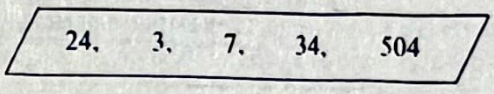
Línea 5. Esta le indica al computador que evalúe la expresión aritmética de la derecha, sumando 24 + 3 + 7, y que almacene el resultado, que es 34, en una localización de memoria, con nombre LS:

LS ← 34

Línea 6. Esta le dice al computador que evalúe la expresión aritmética de la derecha, multiplicando 24 por 3 por 7, y que almacene el resultado en LP:

LP ← 504

Línea 7. Esta le indica al computador que imprima en su orden los valores almacenados en I, J, K, LS, LP. Para representar esto, encerramos la salida en un paralelogramo como sigue:



(Véase *comentario* al final de la sección sobre esta proposición PRINT sin formato.)

Línea 8. Esta le dice al computador que finalice la ejecución.

Línea 9. Cualquier programa debe completarse con una proposición END.

Salida

Supondremos que nuestro dispositivo de salida es una impresora de líneas. Se supone que el papel para imprimir, de la impresora de líneas, tiene 132 columnas (espacios para imprimir). Normalmente, se utiliza una proposición FORMAT para indicarle al computador dónde debe colocar la salida en el papel de imprimir. En este capítulo, utilizamos una proposición PRINT sin formato, de tal manera que hay un formato preasignado para la salida. La salida aparecerá en una línea, como se muestra en la figura 1-5.

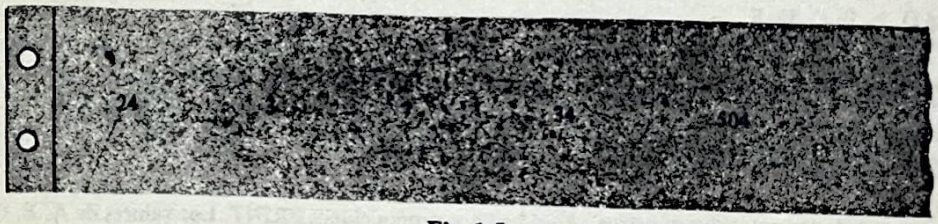


Fig. 1-5

Comentario. Si un lector corre nuestro programa en un computador que no acepte proposiciones PRINT sin formato, debe entonces reemplazar la proposición PRINT (línea 7), por las dos proposiciones siguientes:

```
      WRITE(6, 31) I, J, K, LS, LP
31    FORMAT(5I12)
```

(La pareja WRITE-FORMAT se explicará en el capítulo 3.)

1.5 LECTURA DE DATOS

El programa anterior genera sus propios datos durante su ejecución, ya que explícitamente asigna valores a I, J y K. Para un computador, también es posible "leer" los datos de tarjetas perforadas, cintas magnéticas o de algún otro dispositivo de entrada. Por lo tanto, suponemos que se utiliza una lectora de tarjetas; el computador leerá los datos de un paquete de tarjetas. (Como se ilustró en la figura 1-3, el paquete de datos aparece después del programa Fortran en el paquete Fortran dado al computador.)

La proposición READ se utiliza para instruir al computador acerca de cómo debe tomar los datos de una tarjeta perforada. Una típica proposición READ sin formato, es como sigue:

READ, I, J, K, M, N

Observe que la palabra READ, lo mismo que cada variable, excepto la última, está seguida por una coma. Esta proposición le indica al computador que almacene los cinco primeros valores de la tarjeta de datos en las localizaciones I, J, K, M, N respectivamente.

La principal ventaja al utilizar la proposición READ es hacer un programa más general e independiente de los datos de entrada. Por ejemplo, si en el programa anterior se hubiese utilizado una proposición READ para leer los valores I, J, K, entonces ese programa se podría utilizar para calcular la suma y el producto de cualesquiera valores de I, J y K y no solamente para los valores 24, 3 y 7.

EJEMPLO 1.1

(*a*) Reescribimos el programa anterior pero ahora leeremos los valores de I, J y K. Específicamente, las tres proposiciones después del comentario, se reemplazan por una sola proposición.

```
READ, I, J, K
```

y la tarjeta de datos se perfora como la tarjeta A de la figura 1-6. Si el lector no tiene acceso a la entrada/salida sin formato, debe utilizar las dos proposiciones

```
      READ(5, 10) I, J, K
10    FORMAT(3I5)
```

y perforar una tarjeta de datos, como la tarjeta B de la figura 1-6. La salida de este programa será la misma que la del primer programa.

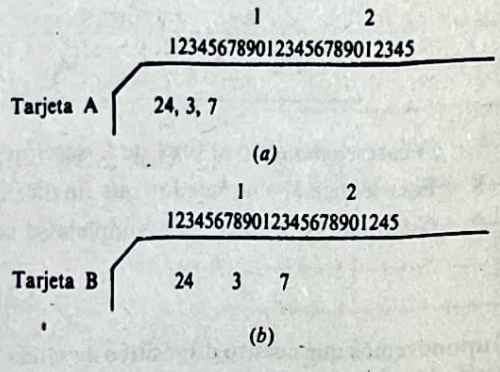


Fig. 1-6

(*b*) Explique el propósito del programa

```
READ, A, B, C. D, E
SUM = A + B + C + D + E
PRINT, A, B, C, D, E
PRINT, SUM
STOP
END
```

Este programa calcula la suma de cinco números. Aquí hay dos proposiciones PRINT. Los valores de A, B, C, D, E se imprimen en una línea y su suma, en una segunda línea.

(*c*) Explique el propósito del programa

```
READ, B, H
AREA = (B*H)/2.0
PRINT, B, H, AREA
STOP
END
```

Este programa calcula el AREA de un triángulo con base B y altura H. Aquí B, H y AREA se imprimen en una línea.

1.6 EJECUCION DE DECISIONES

Un computador normalmente ejecuta las proposiciones de un programa una tras otra, como en los programas anteriores. Sin embargo, algunas veces deseamos que el computador repita o salte un segmento de un programa, es decir, que el computador vaya de un punto del programa a otro. Esto se puede lograr por medio de la proposición GO TO, la cual tiene la forma

```
GO TO n
```

donde *n* es un *número de proposición*. Esta proposición le indica al computador que transfiera el control a la proposición rotulada como *n*; esto es, la siguiente proposición que queremos ejecutar es la rotulada como *n*.

Uno de los aspectos más importantes de un computador es su capacidad para comparar valores y ejecutar decisiones. Por ejemplo, consideremos la proposición Fortran

```
IF(N.LT.300) GO TO 100
```

(En Fortran la notación ".LT." significa "menor que".) Esta proposición IF instruye al computador como sigue:

> si $n < 300$ es verdadero, entonces ejecute la proposición numerada como 100; de lo contrario continúe con la siguiente proposición después de la proposición IF.

En otras palabras, la proposición hace la pregunta "¿Es N < 300?" Si la respuesta es sí, entonces el computador transfiere el control a la proposición rotulada como 100; si la respuesta es no, entonces el computador continúa con la siguiente instrucción.

El siguiente programa ilustra estos conceptos. (Tales conceptos se estudian con detalle en el capítulo 4.) El programa instruye al computador para que imprima los enteros impares positivos de 1 a 11, así como sus cuadrados.

```
C     PROGRAMA PARA IMPRIMIR ENTEROS IMPARES Y SUS CUADRADOS
      N = 1
   21 K = N*N
      PRINT, N, K
      N = N + 2
      IF(N.LT.12) GO TO 21
      STOP
      END
```

Antes de discutir la ejecución del programa, notamos que la proposición

```
N = N + 2
```

puede parecer paradójica. De la sección 1.4 sabemos, sin embargo, que en Fortran el signo = no representa igualdad, sino que denota una proposición de asignación (que incluye dos acciones). El computador primero evalúa N + 2 utilizando el valor actual de N, y luego almacena este nuevo número en la localización N. En otras palabras, la proposición N = N + 2 dice que el nuevo valor de N es su valor actual más 2.

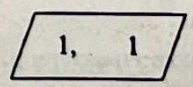
Ejecución del programa

Paso

1. Línea 1. Este es un comentario, ya que se ha perforado la letra C en la primera columna.
2. Línea 2. $N \leftarrow 1$
3. Línea 3. Esta le indica al computador que multiplique el valor de N por sí mismo y que almacene el resultado en K:

   $K \leftarrow 1$

4. Línea 4. Se imprime el valor de N y K:

5. Línea 5. Esta le dice al computador que sume 2 al valor actual de N y la suma (cuyo valor es 3), la almacene en la localización N. (De este modo el valor anterior de N se borra.) Representemos el hecho de que el valor 1 de N se borra y se sustituye por el valor 3 por

   $N \leftarrow \not 1\ 3$

6. Línea 6. Esta hace la siguiente pregunta:

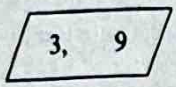
   ¿N es menor que 12?

   Puesto que N = 3, la respuesta es sí, esto es, la expresión entre paréntesis es verdadera. De ahí que el computador va a la proposición numerada como 21, es decir, a la línea 3 del programa.

7. Línea 3. Eleva al cuadrado al valor actual de N y almacena su resultado, el cual es 9, en la localización K. (De esta manera, el valor anterior de K se borra.)

   K ← 9

8. Línea 4. Imprime los valores actuales de N y K:

9. Línea 5. El computador aumenta en 2 el valor actual de N; de ahí que ahora tenemos

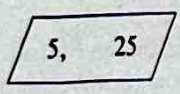
   N ← 5

10. Línea 6. La respuesta a la pregunta "¿N es menor que 12?" es sí, por lo tanto, el computador retorna a la proposición numerada como 21, o sea, a la línea 3 del programa.

11. Línea 3. Eleva al cuadrado el valor actual de N y almacena el resultado, que es 25, en K:

    K ← 25

12. Línea 4. Imprime los valores actuales de N y K:

13. Línea 5. El computador incrementa el valor de N en 2:

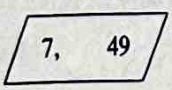
    N ← 7

14. Línea 6. Puesto que 7 < 12, el computador regresa a la línea 3.

15. Línea 3. El computador eleva al cuadrado el valor de N y su resultado, lo almacena en K:

    K ← 49

16. Línea 4. El computador imprime los valores de N y K:

17. Línea 5. El computador incrementa el valor de N en 2:

    N ← 9

18. Línea 6. Puesto que 9 < 12, el computador regresa a la línea 3.

19. Línea 3. El computador eleva al cuadrado el valor de N y este entero lo almacena en K:

    K ← 81

20. Línea 4. El computador imprime los valores de N y K:

9, 81

21. Línea 5. El valor de N se incrementa en 2:

N ← ~~1~~ ~~3~~ ~~5~~ ~~7~~ ~~9~~ 11

22. Línea 6. Puesto que 11 < 12, el computador retorna a la línea 3.

23. Línea 3. El valor actual de K se reemplaza por el valor de N elevado al cuadrado.

K ← ~~1~~ ~~3~~ ~~25~~ ~~49~~ ~~81~~ 121

24. Línea 4. Se imprimen los valores de N y K:

11, 121

25. Línea 5. El valor de N se incrementa en 2:

N ← ~~1~~ ~~3~~ ~~5~~ ~~7~~ ~~9~~ ~~11~~ 13

26. Línea 6. La respuesta a la pregunta "¿N es menor que 12?" es ahora negativa; esto es, la expresión encerrada entre paréntesis es falsa. Por lo tanto, el computador va a la siguiente instrucción del programa.

27. Línea 7. Esta le indica al computador que PARE.

Salida

Cada vez que se encuentre la proposición PRINT, el computador imprime la salida en otra línea. De ahí que cada número impar y su cuadrado aparecerán en líneas separadas, como se ve en la figura 1-7.

Comentario 1. Si la línea 6 del programa se reemplaza, digamos por

```
IF(N.LT.100) GO TO 21
```

entonces el computador imprimiría los enteros impares y sus cuadrados, desde 1 hasta 99.

Comentario 2. Si un lector corre nuestro programa en un computador que no acepte proposiciones PRINT sin formato, entonces él debe reemplazar la proposición PRINT sin formato, por las dos siguientes proposiciones:

```
      WRITE(6, 31) N, K
31    FORMAT(2I15)
```

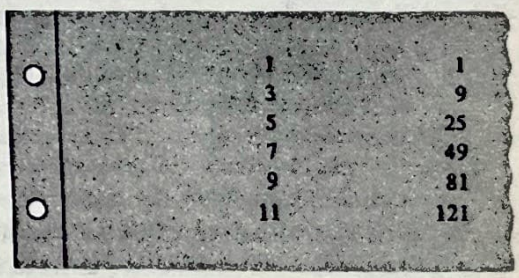


Fig. 1-7

1.8 RESEÑA DE LOS COMPUTADORES Y DE LOS LENGUAJES

Esta sección da una reseña sobre los computadores y los lenguajes de computador. Aunque el lector puede tocar ligeramente esta sección con una primera lectura, hay ciertos términos y detalles, por ejemplo, las proposiciones ejecutables y no ejecutables, que se requieren para aclarar la comprensión de los temas subsiguientes. Aquí hemos incluido todo el material complementario, y los lectores pueden utilizar esta sección como referencia cuando lo estimen necesario.

(*a*) Unidades del computador

Un computador consta de cinco componentes principales: la unidad de entrada, la unidad de salida, la unidad de memoria, la unidad de control y la unidad aritmética. Estas unidades se muestran en la figura 1-8.

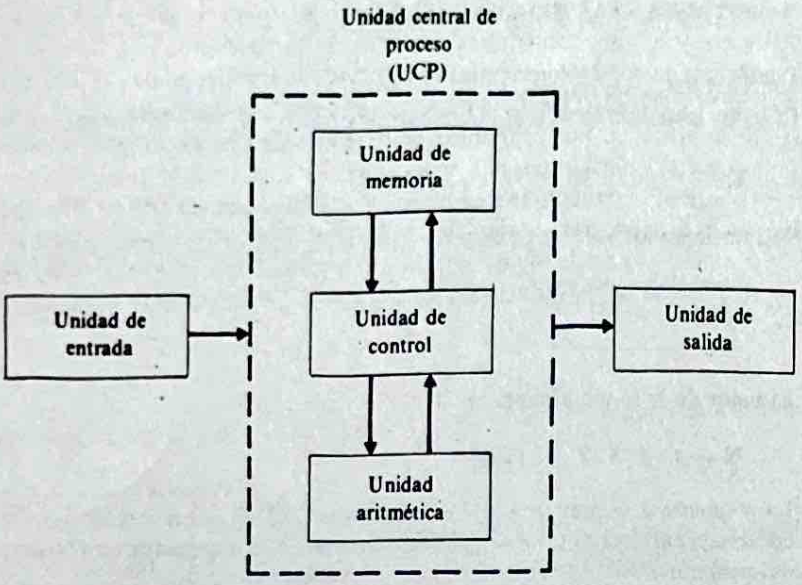


Fig. 1-8

Las unidades de entrada y salida son dispositivos que permiten al computador recibir o exhibir la información. Ellas pueden ser lectoras de tarjetas, cintas magnéticas, lectoras de cintas de papel, impresoras de líneas, TRC (tubos de rayos catódicos) terminales, o consolas con máquinas de escribir. Algunas de estas pueden utilizarse tanto en la entrada como en la salida.

Las tres unidades (unidad de memoria, unidad de control y la unidad aritmética) integran la llamada *Unidad Central de Proceso (UCP).* Estas tres unidades tienen el siguiente propósito:

1. La *unidad de memoria* es el componente donde se almacenan todos los datos y resultados. En la actualidad, los computadores se denominan *computadores de programa almacenado*, puesto que los programas en sí también se almacenan en la memoria. La unidad de memoria consiste de muchas, muchísimas celdas, cada una de las cuales es capaz de almacenar una unidad de información (palabra). Estas celdas de naturaleza electromagnética, se llaman también *localización de almacenamiento (variables).* Para distinguirlas, se numeran secuencialmente. A menudo, estos números hacen referencia a *direcciones* de memoria. Para el usuario, estas direcciones se dan en términos de símbolos (nombres).
2. La *unidad de control* coordina todas las actividades de los varios componentes del computador. Ella envía comandos y señales de control y determina la secuencia de las diferentes instrucciones.
3. La *unidad aritmética* consta de circuitos eléctricos que desarrollan las distintas operaciones aritméticas y lógicas.

(*b*) Jerarquía de los lenguajes de programación

Cualquier computador tiene su propio lenguaje de máquina. Las instrucciones para el computador deben darse en este lenguaje, puesto que éste es el único lenguaje que, realmente entiende un computador. Sin embargo, este tipo de lenguaje varía de máquina a máquina, y es una función del equipo del computador. Los programas escritos en lenguajes de máquina dependen de ella y, por lo tanto, son útiles solamente para cada máquina en particular.

Las instrucciones en lenguaje de máquina se representan usualmente por números binarios, es decir, secuencias que contienen 1's y 0's; la programación en lenguaje de máquina es muy complicada y detallada.

La mayor prioridad en la escala (jerarquía) de los lenguajes de programación la tienen los *lenguajes de ensamble*. Aquí, las secuencias de 1's y 0's se reemplazan por símbolos, de tal suerte que las instrucciones se pueden dar en códigos simbólicos llamados *nemotécnicos*. Un programa *ensamblador* es entonces necesario en el computador, para traducir los programas en lenguaje de ensamble, a instrucciones en lenguaje de máquina. Puesto que la estructura de los lenguajes de ensamble es muy similar a la del lenguaje de máquina, los programadores deberán tener en cuenta varios detalles, tales como el indexado y las localizaciones de almacenamiento, para poder escribir secuencias complejas de instrucciones.

Actualmente, podemos elaborar programas de computador en un lenguaje "casi escrito en inglés", tal como el FORTRAN, el ALGOL, el COBOL, el PL/I, etc. Estos son llamados *lenguajes compiladores* y son casi independientes de la máquina. Los programas escritos en este tipo de lenguaje pueden correrse, con modificaciones muy pequeñas, en máquinas diferentes, mientras exista un compilador para dicha máquina. Un *compilador* es un programa especial que traduce los programas escritos en este tipo de lenguaje de alto nivel, a lenguaje de máquina. El programa original se denomina el *programa fuente* y su traducción recibe el nombre de *programa objeto*. (Véase la figura 1-9.) Es importante recordar que un programa escrito en lenguaje de alto nivel, primero debe traducirse a instrucciones en lenguaje de máquina, antes de que el computador los ejecute.

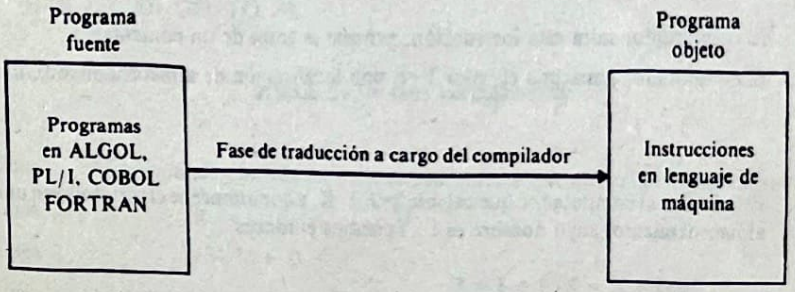


Fig. 1-9

Aunque los lenguajes compiladores son menos eficientes que los lenguajes de máquina y que los lenguajes ensamblados, ellos alivian a los programadores del cuidado de tener en cuenta detalles minuciosos, tales como las localizaciones de almacenamiento, etc. Por otro lado, son mucho más fáciles de aprender y utilizar.

Como se estableció anteriormente, el lenguaje de alto nivel que aprenderemos se denomina Fortran. Utilizaremos el "término" computador en su sentido más amplio, es decir, que incluirá el compilador, la impresora de líneas y otros accesorios.

(*c*) Proposiciones ejecutables y no ejecutables

Como se anotaba antes, nuestro programa Fortran será manipulado por un compilador Fortran, que traduce el programa a instrucciones en lenguaje de máquina. Algunas de las proposiciones Fortran se utilizan solamente para proporcionar información al compilador; ellas no generan instrucciones en lenguaje de máquina. Estas proposiciones reciben el nombre de *proposiciones no ejecutables*. En otras palabras, las *proposiciones ejecutables* son aquéllas que se traducen en alguna clase de instrucciones en lenguaje de máquina.

La proposición STOP es una proposición ejecutable; su equivalente en lenguaje de máquina hace que el computador deje de ejecutar la siguiente instrucción del programa en lenguaje de máquina. Por otra parte, la proposición END es una proposición no ejecutable. Ella le indica al compilador que es la proposición final del programa y que no hay más proposiciones Fortran para traducir a instrucciones en lenguaje de máquina. Puesto que todo el programa Fortran se traduce antes de que cualquier instrucción sencilla en lenguaje de máquina se ejecute, esto explica la razón para que siempre la proposición END sea físicamente la última proposición de cualquier programa Fortran.

Hay muchas otras proposiciones Fortran no ejecutables. Las mencionaremos a medida que progresemos en el desarrollo del texto.

Problemas resueltos

1.1 Determine la salida del siguiente programa.

```
C       PRIMER PROGRAMA RESUELTO
        J = 1
        K = 3
        L = 2*J + K
        J = 3*J + 2*L
        K = K + 2
        L = J + K + L
        PRINT, J, K, L
        STOP
        END
```

Ejecución del programa

Línea 1. El computador salta esta instrucción, porque se trata de un comentario.

Línea 2. El computador almacena el valor 1 en una localización de almacenamiento, llamada J:

$$J \leftarrow 1$$

Línea 3.

$$K \leftarrow 3$$

Línea 4. Esta le dice al computador que calcule $2*J + K$, y que almacene el resultado en una localización de almacenamiento, cuyo nombre es L. Tenemos entonces

$$2*J + K = 2 \cdot 1 + 3 = 5$$

De aquí que el computador almacene el valor 5 en L:

$$L \leftarrow 5$$

Línea 5. Esta le indica al computador que calcule $3*J + 2*L$, utilizando los valores actuales de J y L y que sustituya el valor actual de J por este resultado. Tenemos entonces

$$3*J + 2*L = 3 \cdot 1 + 2 \cdot 5 = 13$$

De aquí, que el computador borre el valor actual de J y en su lugar almacene el valor 13. Indicamos esto de la siguiente manera

$$J \leftarrow \not{1} \; 13$$

Línea 6. Esta le dice al computador que incremente el valor de K en 2:

$$K \leftarrow \not{3} \; 5$$

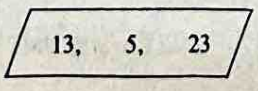
Línea 7. El computador suma los valores actuales de J, K y L:

$$J + K + L = 13 + 5 + 5 = 23$$

Ahora el computador borra el valor actual de L y lo sustituye por el valor 23:

$$L \leftarrow \not{5} \; 23$$

Línea 8. El computador imprime los valores actuales de J, K y L, esto es,



Línea 9. Esta le indica al computador que PARE.

Salida

La salida consiste en los números 13, 5 y 23, los que se imprimen en una línea, ya que la proposición PRINT se utilizó una sola vez.

1.2 Determine la salida del siguiente programa.

```
C       SUMA DE CUADRADOS
        K = 1
        JSUM = 0
   31   JSUM = JSUM + K**2
        K = K + 3
        IF(K.LT.10) GO TO 31
        PRINT, JSUM
        STOP
        END
```

Determine la salida del programa si la línea 6 se reemplaza por

```
IF(K.LT.20) GO TO 31
```

Ejecución del programa

Paso

1. Línea 1. El computador salta esta instrucción, puesto que ella es un comentario.
2. Línea 2. $K \leftarrow 1$
3. Línea 3. $JSUM \leftarrow 0$
4. Línea 4. El computador calcula JSUM + K**2 (donde K**2 es la representación FORTRAN para K^2), utilizando los valores actuales de JSUM y K:

   $$JSUM + K**2 = 0 + 1^2 = 1$$

   Ahora el computador borra el valor 0 en JSUM y lo sustituye por el valor 1:

   $JSUM \leftarrow \not{0}\ 1$
5. Línea 5. El computador incrementa el valor de K en 3:

   $K \leftarrow \not{1}\ 4$
6. Línea 6. Hace la pregunta:

   ¿K es menor que 10?

   Puesto que K = 4, la respuesta es sí, esto es, la proposición encerrada entre paréntesis es verdadera. De ahí que el computador va a la proposición numerada como 31, que es la línea 4 en el programa.
7. Línea 4. Le indica al computador que incremente el valor de JSUM en K^2. Como K = 4, el computador suma 16 al valor actual de JSUM:

   $JSUM \leftarrow \not{0}\ \not{1}\ 17$
8. Línea 5. $K \leftarrow \not{1}\ \not{4}\ 7$
9. Línea 6. El computador va a la proposición número 31, que está en la línea 4, porque

   $K < 10$

   es verdadero.
10. Línea 4. Como K = 7, el computador suma 49 al valor actual de JSUM:

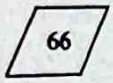
    $JSUM \leftarrow \not{0}\ \not{1}\ \not{6}\ 66$
11. Línea 5. $K \leftarrow \not{1}\ \not{4}\ \not{7}\ 10$

12. **Línea 6.** Puesto que K = 10, la proposición

$$K < 10$$

es falsa; entonces el computador va a la siguiente proposición del programa.

13. **Línea 7.** El computador imprime el valor actual de JSUM, que es



14. **Línea 8.** Esta le dice al computador que PARE.

Salida

La salida está representada solamente por el entero 66, que es el último valor de JSUM. Observe que este valor final de JSUM es la suma

$$JSUM = 1^2 + 4^2 + 7^2$$

esto es, la suma de los cuadrados de los enteros que comienzan con el valor 1 y se van incrementando de 3 en 3, hasta un entero cuyo valor es menor que 10.

En el caso de que la línea 6 sea reemplazada por

```
IF(K.LT.20) GO TO 31
```

entonces la suma continuará hasta un entero menor que 20. O sea, el valor final de JSUM sería

$$JSUM = 1^2 + 4^2 + 7^2 + 10^2 + 13^2 + 16^2 + 19^2 = 952$$

Así que, en este caso, la salida sería el entero 952.

1.3 Explique el propósito de cada programa:

(*a*)
```
READ, R
AREA = 3.1416*R*R
PRINT, R
PRINT, AREA
STOP
END
```

(*b*)
```
READ, A, B, C, D
PROD = A*B*C*D
PRINT, A, B, C, D
PRINT, PROD
STOP
END
```

(*a*) Este programa calcula el AREA de un círculo cuyo radio es R.

(*b*) Este programa calcula el producto PROD de cuatro números A, B, C y D.

Problemas suplementarios

1.4 Encuentre la salida del siguiente programa.

```
C       PRIMER PROGRAMA SUPLEMENTARIO
        I = 3
        J = 2
        K = 2*I + J
        I = I + 3
        J = I + 3*J - K
        K = 2*J + 3*K
        PRINT, I, J, K
        STOP
        END
```

1.5 Determine la salida del siguiente programa.

```
C       SEGUNDO PROGRAMA SUPLEMENTARIO
        K = 3
        JSUM = 0
   51   JSUM = JSUM + K
        K = K + 4
        IF(K.LT.25) GO TO 51
        PRINT, JSUM
        STOP
        END
```

1.6 Encuentre la salida del siguiente programa.

```
C       TERCER PROGRAMA SUPLEMENTARIO
        I  = 1
        J  = 2
   71   K = 2*I + 3*J
        I  = J + K
        J  = I + 2*J
        IF(K.LT.100) GO TO 71
        PRINT, I, J, K
        STOP
        END
```

1.7 Determine la salida del siguiente programa.

```
C       CUARTO PROGRAMA SUPLEMENTARIO
        I = 1
        PRINT, I
        J = 2
        PRINT, J
   41   K = I + J
        PRINT, K
        I = J
        J = K
        IF(K.LT.99) GO TO 41
        STOP
        END
```

1.8 Encuentre la salida del siguiente programa.

```
C       QUINTO PROGRAMA SUPLEMENTARIO
        J  = 1
        K = 3
   11   PRINT, J
   12   J = J + 1
        IF(J.LT.K) GO TO 11
        K = K + 3
        IF(J.LT.20) GO TO 12
        STOP
        END
```

Respuestas a los problemas suplementarios

1.4 Los enteros 6, 4 y 32 se imprimen en una línea.

1.5 Se imprime el entero 78.

1.6 Los enteros 568, 776 y 464 se imprimen en una línea.

1.7 Los enteros 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 se imprimen en líneas diferentes.

1.8 El computador imprime en líneas diferentes los números de 1 hasta 20 a excepción de los múltiplos de 3, esto es, 1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20.

Capítulo 2

Proposiciones aritméticas

2.1 INTRODUCCION

El capítulo primero ilustró cómo un computador lee y ejecuta algunos programas elementales en Fortran. Ahora estudiaremos formalmente el lenguaje Fortran, para que podamos escribir programas que resuelvan problemas.

La siguiente es una lista de los caracteres usados en el lenguaje Fortran:

Dígitos:	0	1	2	3	4	5	6	7	8	9			
Caracteres alfabéticos:	A	B	C	D	E	F	G	H	I	J	K	L	M
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Símbolos:	+	−	*	/	.	,	'	=	$	(	)		

Hay también un espacio en blanco que se reconoce como carácter. Algunas veces denotaremos este carácter por "$_b$", la letra b escrita como subíndice. Los dígitos, en conjunto con el alfabeto, se denominan caracteres *alfanuméricos*. No hay letras minúsculas

Se distingue el dígito 0 (cero) de la letra O escribiendo una diagonal a través de Ø.

2.2 CONSTANTES NUMERICAS (NUMEROS)

En Fortran hay dos tipos diferentes de números, los enteros y los reales. Estrictamente hablando, los enteros son números que se usan para contar y los reales son números que se usan para medir. La aritmética y la representación (interna y externa) de estos tipos de números, son diferentes. Aquí discutiremos solamente la representación externa, es decir, la forma como los escribimos y el computador los imprime. la representación interna, es decir, la forma como el dato se almacena en la memoria, se discutirá en el apéndice A.

Una constante *entera*, también llamada constante *de punto fijo*, es cualquier número con signo o sin signo, sin punto decimal o cualquier otra puntuación. Los números sin signo se toman como positivos. La longitud máxima de cualquier entero normalmente no excede de 9 dígitos, aunque varía de computador a computador. (Los programadores deben verificar las facilidades de su computador, así como las normas que son dependientes de la máquina.)

EJEMPLO 2.1

Las siguientes son constantes enteras:

15 −8 0 12547 −5286

De otra forma, las siguientes no son constantes enteras aceptables:

21.0 (contiene un punto decimal)

−248. (contiene un punto decimal)

12,357 (contiene una coma)

125000000000 (muy grande)

En Fortran una constante *real*, llamada también constante de *punto flotante*, se puede escribir en dos formas diferentes: la forma decimal y la forma exponencial. En ambos casos, hay una secuencia finita de dígitos, con un punto decimal. En efecto, es el punto decimal lo que distingue una constante real de una constante entera.

Forma decimal. En forma decimal, una constante real se escribe como una secuencia finita de dígitos, con un punto decimal, con signo o sin signo. No puede haber otra puntuación. Un número real puede escribirse con cualquier número de cifras significativas; sin embargo, el número de cifras significativas que puede retener el computador, depende de la longitud de la palabra del computador. En general, un computador no retendrá más de siete u ocho dígitos significativos. De nuevo, los lectores podrían consultar sus facilidades de computación para las normas que son dependientes de la máquina.

EJEMPLO 2.2

Las siguientes son constantes reales aceptables:

28.3 −236.0 24. +.0187 −234.182

Las siguientes no son constantes reales aceptables:

4,356.2 (contiene una coma)
−36 (no tiene punto decimal)

Forma exponencial. Una constante real escrita en forma exponencial contiene dos partes. La primera parte es una constante real en forma decimal (es decir, una secuencia finita de dígitos con un punto decimal). La segunda parte empieza con el carácter E, seguido por una constante entera, con signo o sin signo y con, a lo sumo, dos dígitos. La segunda parte se interpreta como el exponente con base 10. Por ejemplo,

23.1E−4

es una constante real, en forma E, que se utiliza para representar 23.1×10^{-4}. Los siguientes son ejemplos de constantes reales aceptables en forma E:

24.123E04	$24.123 \times 10^4 = 241230.0$
−5.36E−3	$-5.36 \times 10^{-3} = -0.00536$
2.6E+2	$2.6 \times 10^2 = 260.0$

Cuando escribimos un número en forma exponencial, no hay restricción en la colocación del punto decimal. Entonces, cada constante real, en forma E, puede escribirse en varias formas. Por ejemplo,

23.1E−4 2.31E−3 0.0000231E2

representan el mismo número y el computador lo acepta así.

Se puede representar siempre un número real en *forma exponencial normalizada*, es decir, de tal modo que el número que precede el carácter E, (llamado la *mantisa*) esté entre

0.1 y 1.0 (ó −0.1 y −1.0)

En efecto, el computador imprimirá, (salida), de esta manera, un número real con exponente.

Como se mencionó anteriormente, el número de cifras significativas retenidas por el computador depende de la máquina utilizada. Más aun, el exponente máximo que se acepta (generalmente limitado a lo sumo a dos dígitos) es también dependiente de la máquina.

Nota. La noción de una constante real en Fortran es diferente de la noción en matemáticas. En efecto, todos los números reales en matemáticas se aproximan por números racionales en computadores.

Aunque 24.0 y 2.4E1 son diferentes en apariencia (externamente), tienen representaciones internas idénticas, puesto que ambos son números reales. (Ver el apéndice A.) De otra forma, 24 y 24.0 tienen representaciones internas diferentes y se tratan diferentemente en Fortran, puesto que uno denota una constante entera y el otro una constante real. (Matemáticamente, representan el mismo número.)

Aún no hemos discutido cómo instruir al computador para leer o imprimir números enteros y/o reales. Esto se discutirá en el capítulo 3.

2.3 NOMBRES DE VARIABLES (NOMBRES DE LAS LOCALIZACIONES DE ALMACENAMIENTO)

El término *variable* en terminología de computadores significa siempre *elemento de memoria*. La característica principal para recordar acerca de los elementos de memoria es: lectura destructiva (entrada), lectura no destructiva (salida). Esto es, si se almacena información en una celda de la memoria, entonces el contenido original (previo) de la celda será destruido, pero si se copia información desde una celda de la memoria, entonces el contenido (original) de la celda permanecerá intacto.

Si una localización de memoria (variable) se usa para almacenar constantes enteras, entonces se denomina una *variable entera*, y si se usa para almacenar constantes reales, se llama una *variable real*.

Para facilitar la referencia, se asignan nombres a las celdas de memoria. En general, un nombre de variable está compuesto por uno a seis caracteres alfanuméricos, de los cuales el primero debe ser un carácter alfabético. Entonces, IAM, A2X3, BAD, DEAN son nombres de variable aceptables, pero las siguientes no lo son:

2XY	No empieza con un carácter alfabético.
A*B6	* no es un carácter alfanumérico.
INTEREST	Contiene más de seis caracteres.

(Algunos computadores pueden aceptar más de seis caracteres para nombres de variables, pero la mayoría tiene seis como su límite superior.)

Fortran también utiliza las siguientes reglas implícitas para nombres de variable: si el primer carácter es una de las seis letras, I, J, K, L, M, N, entonces es un *nombre de variable entera*; pero si el primer carácter es una de las otras letras, esto es, alguna de las siguientes: A,..., H, O, P,..., Z, entonces es un *nombre de variable real*. (Esta regla puede ser obviada por proposiciones de declaración de tipo, discutidas en la sección siguiente.)

EJEMPLO 2.3

Las siguientes son variables enteras aceptables,

MONEY NEXT J4X2 IDIOT

mientras que las siguientes son variables reales aceptables,

ANSWER X21 BUM ENEMY

Nota. Ciertas palabras, tales como READ, WRITE, PRINT, PAUSE, STOP, END, etc., son parte del lenguaje Fortran y, por lo tanto, no son aceptadas como nombres de variable por la mayoría de los computadores.

2.4 PROPOSICIONES DE DECLARACION DE TIPO—ENTERA, REAL

Algunas veces no es deseable seguir las normas implícitas para escribir variables. Por ejemplo, se puede desear usar el nemotécnico (ayuda de memoria) INT para designar una variable real para la rata de interés. Para eliminar la regla implícita, Fortran nos provee de varias proposiciones declarativas de tipo explícito. Una proposición Fortran que empiece con la palabra REAL y continúe con una lista de variables separadas por comas, declara que esas variables son reales. Entonces,

```
REAL INT
```

informa al computador que a través del programa, INT denotará siempre una variable real. Análogamente, una proposición Fortran que empiece con la palabra INTEGER, seguida por una lista de variables separadas por comas, declara a esas variables como enteras.

Estas *proposiciones de declaración de tipo* proveen información al compilador; entonces, no son proposiciones ejecutables y deben aparecer al principio del programa. Otras proposiciones de tipos diferentes de INTEGER y REAL serán introducidas en capítulos posteriores.

EJEMPLO 2.4

Supongamos un programa que se inicia con las proposiciones:

```
INTEGER A, B, SUM, TIME
REAL NUMBER, MIN, MAX
```

así, que a través del programa, A, B, SUM y TIME denotarán variables enteras y NUMBER, MIN y MAX denotarán variables reales.

Algunos programadores creen que es un esfuerzo adicional práctico, listar todas las variables en proposiciones declarativas de tipo, sin tener en cuenta su primera letra. Por ejemplo, se puede escribir

```
REAL NUMBER, MIN, MAX, RATE
```

aunque RATE es real y no es necesario escribirlo explícitamente en una proposición de declaración de tipo.

2.5 OPERACIONES ARITMETICAS—ARITMETICA DE ENTEROS Y DE REALES

La tabla 2-1 da los símbolos para las cinco operaciones aritméticas básicas de Fortran. Resaltamos el hecho de que las expresiones aritméticas

$$a \cdot b \qquad \frac{a}{b} \qquad a^b$$

se escriben respectivamente, A*B, A/B, A**B.

Tabla 2-1

Operación	Símbolo
Suma	+
Resta	−
Multiplicación	*
División	/
Exponenciación	**

Puesto que hay dos tipos de números, enteros y reales, también hay dos tipos de aritméticas, *aritmética real* y *aritmética entera*. Estos dos tipos de aritmética se realizan en forma diferente, por lo que es muy importante tener una clara comprensión de la diferencia.

(*a*) Aritmética entera

Si los operandos son enteros, la aritmética entera se realiza para que *resulte un entero*. Por ejemplo, la aritmética entera es usada para evaluar

$$5 + 3, \qquad 5*3, \qquad \text{y} \qquad 5 - 3$$

resultando los enteros 8, 15 y 2, respectivamente. La división entera en Fortran, por ejemplo, evaluar I/J donde I y J son enteros, también da un resultado entero, la parte entera del cociente. Esto es, la parte fraccional del cociente se borra (*se trunca*), en división entera. Por ejemplo,

5/4 resulta 1, 4/5 resulta 0, y −7/2 resulta −3

Entonces la división entera en Fortran es diferente de la división ordinaria. Sin embargo, esto se puede usar para nuestro provecho, como se ilustra en el ejemplo 2.5.

(*b*) Aritmética real

Si los operandos son reales, entonces la aritmética real se realiza para dar un valor real. Así, la aritmética real se usa para evaluar

5. + 3., 5.*3., y 5. – 3.

resultando los valores reales 8., 15. y 2., respectivamente. La división por aritmética real en Fortran es similar a la división ordinaria. Esto es,

5./4. resulta 1.25, 4./5. resulta 0.8, y –7./2. resulta –3.5

La parte más importante de la discusión anterior es el hecho de que el tipo de los operandos determina el tipo de aritmética (o el *modo de operación*), lo cual conduce al mismo tipo de resultado. Notemos que la exponenciación se trata en forma diferente. Específicamente, sólo los valores positivos reales pueden ser elevados a potencias reales, mientras que cualquier valor, entero o real, puede ser elevado a una potencia entera. Esto se discute posteriormente al final de la sección 2.8.

EJEMPLO 2.5

Aceptando que el uso convencional de los paréntesis en expresiones aritméticas es también aceptable en Fortran, discuta las diferencias entre

(*a*) (N*2)/2 y (*b*) (N/2)*2

También muestre cómo (*b*) puede usarse para determinar si N es par o impar.

Las expresiones algebraicas que ellas representan son

$$\frac{n \cdot 2}{2} \quad \text{y} \quad \frac{n}{2} \cdot 2$$

respectivamente. Matemáticamente, son idénticas. Sin embargo, bajo aritmética entera Fortran no son equivalentes. Específicamente, tenemos lo siguiente:

(*a*) Aquí N*2 se evalúa primero resultando un entero par, entonces, (N*2)/2 tendrá siempre el valor de N.

(*b*) Aquí N/2 se evalúa primero y de esto resulta la parte entera del cociente. De acuerdo con esto, si N es par, (N/2)*2 resulta N, pero si N es impar, (N/2)*2 resulta N ± 1. Por ejemplo, si N = 8, tenemos

(N/2)*2 = 4*2 = 8

pero si N = 11, tenemos

(N/2)*2 = 5*2 = 10

y si N = –9, tenemos

(N/2)*2 = –4*2 = –8

2.6 EXPRESIONES ARITMETICAS

Debemos comprender cómo el computador evalúa las expresiones aritméticas, para poder escribirlas correctamente.

El computador evalúa una *expresión libre de paréntesis* (es decir, una expresión o subexpresión sin paréntesis), utilizando la siguiente tabla de jerarquía convencional:

Primera jerarquía:	Exponenciacion (**)
Segunda jerarquía:	Multiplicación (*) y división (/)
Tercera jerarquía:	Suma (+) y resta (–)

[Nosotros tratamos la suma y la resta de la misma forma que la suma y la resta binaria; esto es, –2.2**2 significa –(2.2*2), lo cual nos resulta –4.84.]

Puesto que hay tres niveles de prioridad, podemos imaginar que el computador explora tres veces de izquierda a derecha una expresión aritmética libre de paréntesis. La primera vez examina para

exponenciación, la segunda para multiplicación y división y, finalmente, examina para suma y resta. Puesto que el computador explora de izquierda a derecha, si dos operaciones tienen la misma jerarquía, la que esté más a la izquierda se realiza primero.

EJEMPLO 2.6

Consideremos la expresión

```
A*B + C/D*F**2
```

El orden de ejecución se indica en la figura 2-1(*a*). En forma equivalente, este orden puede escribirse como un árbol invertido con operandos como nodos, como se muestra en la figura 2-1(*b*). Observe que la expresión representa

$$a \cdot b + \frac{c}{d} \cdot f^2$$

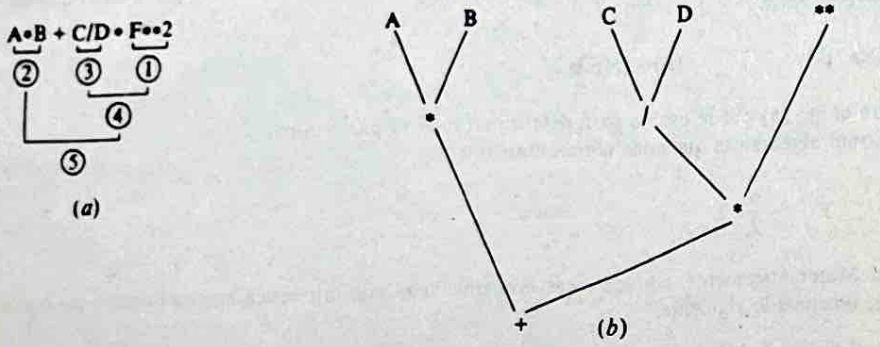
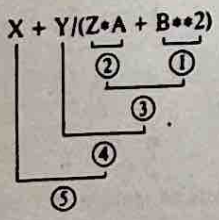


Fig. 2-1

Fortran también permite el uso de paréntesis en la forma convencional. En especial, los paréntesis tienen jerarquía sobre todas las operaciones aritméticas y se trata primero el par más interior de paréntesis.

EJEMPLO 2.7

(*a*) La expresión X + Y/(Z*A + B**2) se evalúa como sigue:



Luego, la expresión algebraica es

$$x + \frac{y}{z \cdot a + b^2}$$

(*b*) La ecuación algebraica

$$x = [(a + b)^2 + (3c)^3]^{a/b}$$

puede escribirse en Fortran como

```
X = ((A + B)**2 + (3.0*C)**3)**(A/B)
```

Es altamente recomendable utilizar paréntesis extras cuando tal uso puede asegurarnos exactitud. Más aún, si una expresión aritmética contiene términos muy grandes y complicados, es siempre deseable romperla en varias proposiciones más cortas que son menos susceptibles a error. Por ejemplo, la ecuación algebraica

$$y = \left[\frac{ab}{c+d} - \frac{g}{5(h+x)}\right]^{1/r}$$

será representada como:

```
T1 = A*B/(C + D)
T2 = G/(5.*(H + X))
 Y = (T1 - T2)**(1./R)
```

Nota: Fortran no permite que dos operandos aritméticos aparezcan uno junto a otro. Entonces, A* – B debe escribirse como A*(– B).

EJEMPLO 2.8

Las siguientes son expresiones aritméticas y sus expresiones correspondientes en Fortran.

	Matemática	Fortran
(*a*)	$a + \frac{b}{c} + d$	A + B/C + D
(*b*)	$\frac{a+b}{c+d}$	(A + B)/(C + D)
(*c*)	$a^3 - b^3$	A**3 – B**3
(*d*)	$\frac{ab}{c-d^2}$	A*B/(C – D**2)
(*e*)	$\frac{ab}{c} - d^2$	A*B/C – D**2
(*f*)	$\frac{a}{bc} - d^2$	A/(B*C) – D**2
(*g*)	$1 + \frac{a}{b + \frac{1}{c}}$	1. + A/(B + 1./C)

Obsérvese que los paréntesis deben aparecer en (*f*), pero no son necesarios en (*e*).

2.7 OPERACIONES CON MEZCLA DE MODOS

Como se discutió en la sección 2.5, cuando una expresión aritmética es evaluada por el computador, el tipo de los operandos determina el tipo de aritmética (modo de operación) que se usa. Entonces, todas las variables y constantes en cualquier expresión aritmética Fortran, deben ser del mismo tipo, es decir, deben ser todas enteras o todas reales. De acuerdo con esto, si

```
A + 3
```

es una expresión donde el operando A es real y el otro entero, se puede uno preguntar: (1) si tal *mezcla de modos* se permite, y (2) aun si es permitida, en qué tipo de aritmética se realizará.

Algunos compiladores Fortran *no* permiten mezcla de modos en las expresiones. En tales casos, se debe tener mucho cuidado en evitar mezclar dos tipos en una expresión. Luego,

A + 3•B deberá escribirse como A + 3.•B
I + 4.0•J deberá escribirse como I + 4•J
B••2 – 4•A•C deberá escribirse como B••2 – 4.•A•C

La elevación de un número real a una potencia entera no se considera una mezcla de modos de operación (ver discusión al final de la sección 2.8), así que B••2 se permite y no necesita cambiarse a B••2.0. (Para la diferencia entre B••2 y B••2.0, ver la sección 2.8 y también el problema 2.17.)

La mayoría de los grandes computadores tiene compiladores que permiten mezclar modos de expresiones. En tales casos, cuando los operandos de una operación son de dos tipos diferentes, es decir, uno entero y el otro real, el entero *se convierte primero* a real y luego se realiza la aritmética real, para conducirnos a un resultado real. Esto puede parecer conveniente, puesto que variables y constantes enteras y variables y constantes reales, pueden aparecer en una expresión; sin embargo, los programadores deberán tener cuidado a fin de no cometer errores cuando aparecen ambos tipos de variables y constantes.

EJEMPLO 2.9

(*a*) Admitiendo que se permite la mezcla de modos aritmética, considere la expresión

(11./2) + 4.3

Puesto que 11. y 2 son de tipos diferentes, el entero 2 se convierte a 2., y se realiza la aritmética real, para dar como resultado el número 5.5. Sumando los números reales 5.5 y 4.3, tenemos 9.8 como resultado.

(*b*) Considere la expresión

(11/2) + 4.3

Se evalúa primero el término 11/2, puesto que 11 y 2 son enteros, 11/2 se evalúa por aritmética entera, resultando 5 (no 5.5). Puesto que los operandos de la suma son de tipos diferentes, el entero 5 se convierte a real y se suma a la constante real 4.3, utilizando aritmética real, lo cual nos conduce al número real 9.3.

Los programadores deberán tener cuidado de que la conveniencia anterior, no llegue a ser "libre". Esto es, cada vez que un valor entero es convertido a real, se utiliza tiempo de máquina. Luego, permitiendo la mezcla de modo en expresiones, aumentaremos el tiempo de ejecución y correremos el peligro de reducir grandemente la eficiencia de ejecución del programa. Además, los programas escritos con mezcla de modos serán menos manejables; es decir, podría ser posible que no se pudieran procesar en diferentes máquinas Finalmente, el uso extensivo de mezcla de modos puede conducir a valores diferentes de los esperados, como se ilustra explícitamente en el ejemplo 2.9.

2.8 FUNCIONES MATEMATICAS INTERNAS

El lenguaje Fortran tiene una biblioteca de funciones matemáticas que pueden ser utilizadas por el programador. Algunas de las más comunes son listadas en la tabla 2-2. Se da una lista mucho más extensa en el apéndice B.

Tabla 2-2

Función	Notación matemática	Fortran
Raíz cuadrada de x	$\sqrt{x}$	SQRT(X)
Valor absoluto de x	$\|x\|$	ABS(X)
Exponencial de x	e^x	EXP(X)
Seno de x	sen x	SIN(X)
Coseno de x	cos x	COS(X)
Log de x	$\log_{10} x$	ALOG10(X)
Log natural de x	$\log_e x$	ALOG(X)
Convertir el entero I a real		FLOAT(I)
Truncar real x en entero		IFIX(X)

Aunque la función de los subprogramas se discutirá en detalle en el capítulo 7, algo de los términos y de la notación se explicará aquí. La expresión

SQRT(X)

denota el valor que la función raíz cuadrada asigna a X. Se llama a X, el *argumento* de la función y SQRT se llama el *nombre* de la función. Obsérvese que el nombre de la función es seguido por un par de paréntesis que encierran el argumento.

La tabla 2-2 utiliza a X y a I como argumentos de funciones. La X indica que el argumento debe ser real, mientras que la I indica que el argumento debe ser de tipo entero. De otra forma, el tipo de un valor de la función se determina por el nombre de la función de acuerdo con las reglas usuales para nombres de variables. Por ejemplo,

ALOG(X) denota un valor real correspondiente al argumento real X

IFIX(X) denota un valor entero correspondiente al argumento real X

FLOAT denota un valor real correspondiente al argumento entero I

y así, sucesivamente.

Las funciones FLOAT e IFIX no son funciones matemáticas usuales. La función FLOAT convierte un valor entero en su valor real equivalente; por ejemplo,

FLOAT(4) da el valor 4.0 y FLOAT(−25) da el valor −25.0

Generalmente la función FLOAT se utiliza para evitar el uso de la mezcla de modos. Por ejemplo, si SUM denota la suma de N números, entonces se puede escribir

PROM = SUM/FLOAT(N)

para obtener el promedio PROM de los N números. La función IFIX convierte por truncamiento un valor real en un valor entero; por ejemplo,

IFIX(2.8) da el valor 2 e IFIX(−3.7) da el valor −3

Resaltamos el hecho de que la función IFIX no redondea los valores reales a su entero más cercano.

Deberán tenerse en cuenta las siguientes reglas:

1. Los argumentos de las funciones no necesitan ser solamente simples variables. Pueden ser cualquier expresión aritmética del tipo requerido. Por ejemplo, la traducción Fortran de

$\sqrt{b^2 - 4ac}$ será SQRT(B**2 − 4.*A*C)

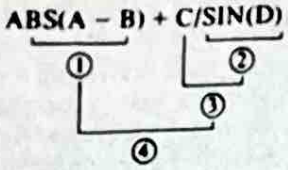
2. Las funciones pueden aparecer como una parte de cualquier expresión aritmética en tanto que sea dado el argumento apropiado. Por ejemplo, la traducción Fortran de

$|a - b| + \frac{c}{\operatorname{sen} d}$ será ABS(A − B) + C/SIN(D)

3. Un argumento de una función puede llamar otra función. Esto es, podemos tener funciones de funciones. Por ejemplo, la traducción Fortran de

$\operatorname{sen} \left|\frac{x - y}{x + y}\right|$ será SIN(ABS((X − Y)/(X + Y)))

4. Para una expresión aritmética libre de paréntesis que involucre funciones de biblioteca, la evaluación de esas funciones tiene precedencia sobre las reglas discutidas en la sección 2.6. Por ejemplo, la expresión



será evaluada en el orden que se indica. Nótese que el argumento de ABS es una expresión; entonces, en su evaluación, las reglas de jerarquía se aplican de nuevo.

EJEMPLO 2.10

Las siguientes son expresiones matemáticas y sus expresiones Fortran correspondientes.

	Matemática	Fortran
(*a*)	$\frac{\operatorname{sen} x}{\lvert y\rvert + \cos z}$	SIN(X)/(ABS(Y) + COS(Z))
(*b*)	$\frac{e^{x+y}}{x+y}$	EXP(X + Y)/(X + Y)
(*c*)	$e^{\lvert x-y\rvert} + x$	EXP(ABS(X − Y)) + X
(*d*)	$a + \frac{b}{\lvert m-n\rvert}$	A + B/ABS(FLOAT(M − N))

Un error común, cometido por muchos programadores principiantes, consiste en la escritura de la constante π. Como π no es una constante de biblioteca, y además π no es un caracter aceptado por Fortran, debe usarse un valor aproximado de π. Por ejemplo, la expresión Fortran para

$\operatorname{sen}(2n\pi + x)$ será SIN(2.*N*3.14 + X)

donde 3.14 es utilizado por π. En la misma forma, podemos asignar 3.14 a PI (ver la sección 2.9), y entonces escribir

SIN(2.*N*PI + X)

Finalizamos esta sección con las siguientes sutilezas, muy importantes, sobre los puntos concernientes a la exponenciación, puesto que ellos comprenden la función logarítmica:

1. Cuando un valor es elevado a una potencia real como en X**Y, se evalúa internamente como

   $e^{y \ln(x)}$

   donde ln es la función de logaritmo natural, esto es, como EXP(Y*ALOG(X)). (Ver la tabla 2.2.) Puesto que el argumento debe ser real y positivo, solamente valores reales positivos de X pueden ser elevados a potencias reales. Los programadores deberán tener especial cuidado con esto, puesto que el error de elevar un valor negativo a una potencia real se detecta solamente en tiempo de ejecución.

2. Cuando un número es elevado a una potencia entera como en X**I, se hace por multiplicación. Entonces, cualquier valor, entero o real, positivo o negativo, puede elevarse a una potencia entera. (También, ver el problema 2.17.)

3. Matemáticamente, a^{b^c} significa $a^{(b^c)}$ lo cual es diferente de $(a^b)^c$. Sin embargo, A**B**C puede ser tratado como (A**B)**C por algunos compiladores y como A**(B**C) por otros. Entonces, recomendamos a los programadores utilizar paréntesis cuando escriban tal expresión. En general, cuando haya dudas, es deseable utilizar paréntesis.

2.9 PROPOSICIONES DE ASIGNACION ARITMETICA

Para que el computador ejecute una expresión aritmética, todas las variables que aparecen en la expresión deben tener valores dados dentro del programa, antes de que se usen; es decir, deben estar *definidas*. La mayoría de los computadores darán un mensaje de error, si cualquier variable en una expresión no está definida. ¿Cómo definimos una variable, es decir, cómo almacenamos un valor en la memoria? Una variable se define mediante la lectura de su valor de un paquete de datos (ver el capítulo 3) o por una proposición aritmética de asignación.

Simbólicamente, denotamos el almacenamiento (asignación) de la constante 1.5 en una localización llamada X por

$$X \leftarrow 1.5$$

Esto se llama una *proposición de asignación*. En Fortran, $\leftarrow$ se representa por = , así que $X \leftarrow 1.5$ se escribe en Fortran como

```
X = 1.5
```

En la misma forma, supongamos que X está realmente definido y que deseamos computar x^2 y almacenar el valor en la localización Y; esto es, $Y \leftarrow x^2$. En Fortran escribimos

```
Y = X**2
```

En general, una *proposición Fortran de asignación aritmética* tiene la siguiente forma:

Nombre de variable = expresión aritmética

Hacemos énfasis en el hecho de que el lado izquierdo de = debe ser el nombre de una variable singular.

Es desafortunado que el símbolo = sea usado para proposiciones de asignación en Fortran, puesto que = se usa convencionalmente para significar igualdad. Sugerimos insistentemente a los lectores que recuerden a = como "asignación", "reemplaza", "copia", etc. En nuestras discusiones, algunas veces utilizaremos el símbolo $\leftarrow$ para asignación en lugar de su equivalente Fortran =, cuando tal hecho favorezca la comprensión.

Nota. Es extremadamente importante saber que en una proposición de asignación aritmética, tienen lugar dos acciones separadas:

1. La computación de la expresión aritmética a la derecha del signo = .
2. La asignación del resultado de (1) a la localización de almacenamiento en la izquierda del signo = .

Entonces, se acepta que los dos lados de = puedan ser de tipos diferentes. Cuando los dos lados de = tienen tipos diferentes, la asignación en el segundo paso incluye convertir el resultado de (1) al mismo tipo de localización de almacenamiento indicado en la izquierda de=. En particular, un valor real será truncado antes de que sea asignado a una localización entera.

Muchos programadores principiantes a menudo tienen una confusión entre mezcla de modos de expresiones y proposiciones de asignación con diferentes tipos en ambos lados. La mezcla de modos se refiere a expresiones aritméticas en las que los operandos son de tipo diferente. Las proposiciones de asignación se refieren a la acción de almacenar un valor en la memoria; es una instrucción Fortran, y puede incluir la evaluación de una expresión aritmética. La expresión, en sí, no es una instrucción. Algunos compiladores no aceptan mezcla de modos en expresiones. Sin embargo, todos los compiladores aceptan proposiciones de asignación con tipos diferentes a los lados de = .

EJEMPLO 2.11

(*a*) Considerar la proposición

```
INCOME = PRIN*(1. + XINT)
```

La expresión aritmética en el lado derecho de = es de tipo real. El resultado de su computación se convierte primero en un entero (por eliminar su parte fraccionaria), antes de que se almacene en INCOME, puesto que es una variable entera.

Específicamente, si PRIN = 100.00 y XINT = 0.0625, entonces el resultado de PRIN*(1. + XINT) es 106.25. Sin embargo, antes de que sea copiado en INCOME, se trunca la fracción .25 y el entero 106 se almacena en INCOME.

(*b*) Considerar la proposición

ASSET = INCOME + LOOT

Aquí la expresión aritmética se calcula en modo entero y luego se convierte a tipo real, antes de que sea almacenada en la localización real llamada ASSET.

(*c*) Considerar la proposición

SUM = SUM + X

Puesto que hay dos acciones separadas en la proposición de asignación, se ejecuta primero SUM + X. Entonces, se suman los valores corrientes de SUM y X. Su resultado se almacena de nuevo en SUM. Efectivamente, la proposición dice:

"El nuevo valor de SUM es el resultado de sumar el valor antiguo de SUM y X".

Obviamente, si fuera interpretado como "igualdad", la proposición dada SUM = SUM + X no tendría ningún sentido.

2.10 ARITMETICA DEL COMPUTADOR

Quedaríamos descontentos si no mencionáramos al lector algunos de los sutiles inconvenientes de la aritmética del computador.

Puesto que un computador puede retener solamente un número particular de cifras significativas (porque la longitud de la palabra es finita), todos los números dentro del computador son números racionales. Así que la mayoría de los números, en particular números irracionales, tales como $\sqrt{2}$, π, etc., no son almacenados con su valor verdadero, sino con aproximaciones racionales. En efecto, muchos números racionales con expansiones decimales determinadas, no tienen expansiones binarias determinadas; luego no se almacenan con su valor verdadero. En la misma forma hay nuevos problemas con las aritméticas del computador. Discutiremos algunos de ellos aquí. (Ver también los problemas 2.17 y 2.18.)

(*a*) Error de conversión

La mayoría de los computadores almacena números en su forma binaria. Así, el número 1/2 que se representa en el sistema decimal por 0.5, se representa en tales computadores por 0.1. Sin embargo, el número 1/10 no tiene una representación binaria finita:

$$1/10 = 0.0001100110011\ldots\ldots$$

Dependiendo de la longitud de la *palabra* (*celda de memoria*), el error de conversión puede ser despreciable. Sin embargo, el error se propagará dependiendo del número de operaciones aritméticas.

(*b*) Propagación del error de redondeo

Supongamos, por ejemplo, que nuestro computador es una máquina decimal que retiene solamente tres cifras significativas. Entonces el número 1/3 se almacena en el computador, como 0.333. Consideremos la suma de 1/3, a sí mismo, por 10 veces. Tenemos:

$$\begin{array}{r} .333 \\ +\ .333 \\ \hline .666 \\ +\ .333 \\ \hline .999 \\ +\ .333 \\ \hline 1.332 \end{array}$$

En este punto de la suma, el computador solamente retiene 1.33, puesto que hemos tomado una máquina de tres dígitos. Continuando, obtenemos:

$$
\begin{array}{r}
1.33 \\
+\ .333 \\
\hline
1.66 \\
+\ .333 \\
\hline
1.99 \\
+\ .333 \\
\hline
2.32 \\
+\ .333 \\
\hline
2.65 \\
+\ .333 \\
\hline
2.98 \\
+\ .333 \\
\hline
3.31
\end{array}
$$

En lugar de obtener el resultado teórico 3.33, obtenemos 3.31, y el error se propaga ahora más rápidamente. En efecto, si se suma .333 a sí mismo 300 veces, en nuestra máquina, obtiene solamente 90.9, en lugar del resultado deseado: 99.9. Más aún, si suma .333 a 100, el resultado será siempre 100. Entonces, al sumar .333 a sí mismo sucesivamente, se llegará con el tiempo sólo hasta 100.

Obviamente, algo se puede hacer en este caso particular, pero los errores se propagarán, si no tenemos cuidado con ellos.

(*c*) No asociatividad de la aritmética

Considerar las expresiones en Fortran equivalentes matemáticamente

$$(A + B) + C \qquad \text{y} \qquad A + (B + C)$$

Supongamos que el computador es exacto para ocho cifras significativas y que

$$A = 0.30000000 \qquad B = 87654321. \qquad C = -87654322.$$

Puesto que el computador puede solamente retener ocho dígitos,

$$A + B = 87654321.$$

y A no afecta el resultado de A + B. Luego,

$$(A + B) + C = 87654321. - 87654322. = -1.0000000$$

De otra forma, B + C = − 1.0000000; entonces,

$$A + (B + C) = 0.30000000 - 1.0000000 = -0.70000000$$

Hay un 30% de diferencia en el resultado y esto nos puede conducir a grandes diferencias si esos valores se usan en otros cálculos; es decir, el error de redondeo puede propagarse.

Problemas resueltos

CONSTANTES

2.1 Escribir los siguientes números como constantes enteras en Fortran:

(*a*) 34,275 (*d*) 4.8×10^4 (*g*) −7.3
(*b*) +275 (*e*) −1,278000 (*h*) 12
(*c*) −23.0 (*f*) 2.137×10^2 (*i*) −138

(*a*) 34275 (*d*) 48000 (*g*) Imposible—no es entera
(*b*) +275 (*e*) −1278000 (*h*) 12
(*c*) −23 (*f*) Imposible—no es entera (*i*) −138

2.2 ¿Por qué los siguientes números son inaceptables como constantes enteras en Fortran?

(*a*) 2,371 (*b*) −47.0 (*c*) 28E3 (*d*) 1234500000

(*a*) Contiene una coma.
(*b*) Contiene un punto decimal.
(*c*) Un entero no puede escribirse en forma exponencial.
(*d*) Para la mayoría de los computadores, un entero no puede tener más de nueve caracteres.

2.3 Escribir los siguientes números como constantes reales en la forma dada y en la forma exponencial normalizada:

(*a*) 123. (*d*) 5.63×10^{-8} (*g*) 0.123
(*b*) 12.3 (*e*) −1,234,000 (*h*) −0.356
(*c*) −3,400 (*f*) 0.000347 (*i*) 3×10^{13}

(*a*) 123. ó .123E3 (*d*) 5.63E−8 ó .563E−7 (*g*) 0.123 ó 0.123E0
(*b*) 12.3 ó .123E2 (*e*) −1234000. ó −.1234E7 (*h*) −.356 ó −.356E0
(*c*) −3400. ó −.34E4 (*f*) .000347 ó .347E−3 (*i*) 3.E13 ó .3E14

Hacemos énfasis en el hecho de que una constante real puede escribirse de muchas maneras diferentes en forma exponencial, pero solamente de una manera en forma exponencial normalizada, es decir, como una potencia de 10 veces un número entre 0.1 y 1.0 ó entre −0.1 y −1.0.

2.4 ¿Por qué los siguientes números son inaceptables como constantes reales en Fortran?

(*a*) 3,248.6 (*c*) 1.6E128 (*e*) 4E−2
(*b*) −13 (*d*) E+3 (*f*) 4.3E3.7

(*a*) Contiene una coma.
(*b*) No tiene punto decimal.
(*c*) Un exponente no puede tener más de dos dígitos.
(*d*) Un exponente no puede aparecer solo (se puede escribir 0.0E + 3).
(*e*) El 4 debe tener un punto decimal.
(*f*) La mayoría de los computadores solamente permite exponentes enteros.

VARIABLES

2.5 ¿Cuáles de los siguientes nombres son inaceptables como variables enteras, y por qué?

(*a*) NEXT (*c*) ALPHA (*e*) J + 329 (*g*) 2I45
(*b*) FIRST (*d*) L124 (*f*) MACBETH (*h*) N(3)M

(*a*) Aceptable.
(*b*) Inaceptable—la primera letra no es I, J, K, L, M ó N.
(*c*) Inaceptable—la primera letra no es I, J, K, L, M ó N.
(*d*) Aceptable.
(*e*) Inaceptable—el signo (+) más no es permitido.
(*f*) Inaceptable—tiene más de seis caracteres.
(*g*) Inaceptable—el primer caracter debe ser una letra.
(*h*) Inaceptable—los caracteres deben ser letras o dígitos.

2.6 ¿Cuáles de los siguientes nombres son inaceptables como variables reales? ¿Por qué?

(*a*) ANSWER	(*c*) ROOT2	(*e*) ALTITUDE	(*g*) STOP
(*b*) LAMBDA	(*d*) MAX	(*f*) 4XYZ	(*h*) X-RAY

(*a*) Aceptable.
(*b*) Inaceptable—la primera letra L es reservada para variables enteras.
(*c*) Aceptable.
(*d*) Inaceptable—la primera letra M se reserva para variables enteras.
(*e*) Inaceptable—tiene más de seis caracteres.
(*f*) Inaceptable—el primer carácter debe ser una letra.
(*g*) Inaceptable—es parte del repertorio de Fortran.
(*h*) Inaceptable—el guión (-) no es una letra o dígito.

EXPRESIONES Y OPERACIONES ARITMETICAS

2.7 Suponiendo que la mezcla de modos es inaceptable para el computador, determinar entre las siguientes cuáles son aceptables en Fortran, y encontrar sus valores.

(*a*) 2.4 + 3.85	(*c*) 3*4.5	(*e*) 3* + 5	(*g*) 0 + 4.6
(*b*) 12/5	(*d*) 3.**2	(*f*) 4.3 − 7.93	(*h*) −7/3

(*a*) 6.25.
(*b*) 2 (la parte fraccional se trunca en la división entera).
(*c*) Inaceptable—los operandos tienen tipos diferentes.
(*d*) 9. (los exponentes pueden ser enteros aun cuando la base puede ser real.)
(*e*) Inaceptable—los operandos * y + no pueden aparecer uno junto a otro.
(*f*) − 3.63.
(*g*) Inaceptable—los operandos tienen tipos diferentes.
(*h*) − 2 (la parte fraccional se trunca en la división entera).

2.8 Admitiendo que la mezcla de modos es aceptable, encontrar el valor de cada expresión.

(*a*) 3 + 4.8*2	(*c*) 2.*6/5	(*e*) 7/(4*2.)	(*g*) 2*(3**2)
(*b*) 5.2 + 12/8	(*d*) 2*6/5	(*f*) 7/(4*2)	(*h*) 2*3**2

(*a*) La multiplicación tiene prioridad sobre la suma. Entonces,

4.8*2 resulta 9.6 y 3 + 9.6 resulta 12.6

(*b*) La división entera 12/8 resulta 1, y 5.2 + 1 resulta 6.2.
(*c*) 2.*6 da el número real 12. y 12.5/5 da el número real 2.4.
(*d*) 2*6 da el número entero 12, y la división entera 12/5 da 2 como resultado.
(*e*) 4*2. da el número real 8., y 7/8. da 0.875.
(*f*) 4*2 da el entero 8, y la división entera 7/8 da 0 como resultado.
(*g*) 18.
(*h*) La exponenciación tiene prioridad sobre la multiplicación; luego aún obtenemos 18 como en (*g*).

2.9 Escribir la expresión Fortran correspondiente a cada expresión matemática:

(*a*) $(x + y)(u + v)$ (*d*) $a + \frac{b}{c^2}$ (*f*) $\frac{x^4}{4!}$

(*b*) $3(x + y)$

(*c*) $3xy^2 - 2x^2y$ (*e*) $\frac{a + b}{c \cdot d}$

(*a*) (X + Y)*(U + V) (*c*) 3.0*X*Y**2 – 2.0*X**2*Y (*e*) (A + B)/(C*D)
(*b*) 3.0*(X + Y) (*d*) A + B/C**2 (*f*) X**4/(4.*3.*2.*1.)

2.10 Para cada una de las siguientes expresiones Fortran, encontrar una expresión equivalente, es decir, una expresión que no altere el orden de las operaciones, eliminando paréntesis superfluos.

(*a*) (A*B)*(C + D) (*c*) (A*B*C)/((X*Y)**2)
(*b*) (A*(B**2))/(C*D) (*d*) A*(B*C)

(*a*) A*B*(C + D)
(*b*) A*B**2/(C*D)
(*c*) A*B*C/(X*Y)**2
(*d*) A*(B*C); esto es, ningún paréntesis puede ser eliminado. [Aunque A*B*C, evaluado como (A*B)*C, es matemáticamente equivalente a A*(B*C), el orden de las operaciones es diferente y, así, el error de redondeo puede conducir a resultados diferentes. Ver la sección 2.10.]

2.11 Usando las funciones matemáticas de la tabla 2-2 (página 26), escribir la expresión Fortran correspondiente a cada expresión matemática:

(*a*) $\sqrt{a^2 + b^2}$ (*c*) $\frac{1}{|a \cdot b|} + c$

(*b*) $e^{x+y} - \text{sen}(x + ny)$ (*d*) $\cos(\log_{10}(a + 3b))$

(*a*) SQRT(A**2 + B**2)
(*b*) EXP(X + Y) – SIN(X + N*Y) ó
si el computador no acepta expresiones con mezcla de modos
EXP(X + Y) – SIN(X + FLOAT(N)*Y)
(*c*) 1./ABS(A*B) + C
(*d*) COS(ALOG10(A + 3.0*B))

2.12 Las siguientes son expresiones matemáticas y sus correspondientes expresiones incorrectas en Fortran. Escribir las expresiones correctas en Fortran.

(*a*) $\frac{a \cdot b}{c \cdot d \cdot e}$, AB/CDE (*c*) $\left(\frac{a}{b + c}\right)^2$, A/(B + C)**2

(*b*) $\left(\frac{x}{y}\right)^{n+1}$, (X/Y)**N + 1 (*d*) $\sqrt{\frac{a^2}{b + c}}$, SQRT(A**2/(B + C)

(*a*) A*B/(C*D*E) (*b*) (X/Y)**(N + 1) (*c*) (A/(B + C))**2 (*d*) SQRT(A**2/(B + C))

PROPOSICIONES ARITMETICAS

2.13 Considerar las siguientes proposiciones:

(*a*) A = B + C (*c*) A = A + A (*e*) AB = CD
(*b*) B + C = A (*d*) A + B = C + D (*f*) A = ABS(A)

¿Cuáles de las anteriores, no son aceptables como proposiciones Fortran? Para las que son aceptables, ¿qué condición implícita se supone?

Recalcamos que solamente una variable, es decir, el nombre de una localización de almacenamiento, puede aparecer al lado izquierdo de un signo igual.

(*a*) Aceptable.
(*b*) Inaceptable, puesto que B + C no es una variable.
(*c*) Aceptable.
(*d*) Inaceptable, puesto que A + B no es una variable.
(*e*) Aceptable.
(*f*) Aceptable.

Estamos admitiendo implícitamente que todas las variables a la derecha del signo igual han sido definidas, es decir, esos números están realmente en las localizaciones de almacenamiento que tienen esos nombres.

2.14 Supongamos que A, B, J y K contienen los siguientes valores: A = 2.7, B = 3.5, J = 3 y K = − 2. Encontrar los valores de X y L después de cada par de proposiciones:

(*a*)
```
X = A + J*K**2 + B
L = A + J*K**2 + B
```

(*b*)
```
X = 5*J/4*K
L = 5*J/4.0*K
```

(*c*)
```
X = K/3*A/2
L = K/3*A/2
```

(*d*)
```
X = ABS(A - J*B)/5
L = ABS(A - J*B)/5
```

(*a*) Primero se realiza la exponenciación, luego la multiplicación y luego la suma; entonces el computador evalúa:

$$A + J*K**2 + B = 2.7 + 3(-2)^2 + 3.5 = 2.7 + 12 + 3.5 = 18.2$$

Puesto que X es una variable real, el computador almacena 18.2 en X; pero como L es una variable entera, el computador almacena 18 (esto es la parte entera de 18.2) en L. Así que, X = 18.2 y L = 18.

(*b*) El computador realiza la multiplicación y la división de izquierda a derecha:

$$5*J/4*K = 15/4*K = 3*K = -6$$
$$5*J/4.0*K = 15/4.0*K = 3.75*K = -7.5$$

Puesto que la división es entera, 15/4 da 3; pero 15/4.0 da 3.75. Como X es real, − 6 se convierte al número real, —6.0 y luego se almacena en X. Como L es una variable entera, —7.5 se trunca en —7 y luego se almacena en L. Entonces, X = − 6.0 y L = − 7.

(*c*) Como la división es entera, K/3 da 0. Entonces, X = 0.0 y L = 0.

(*d*) ABS(A − J*B)/5 = |2.7 − 3(3.5)|/5 = |−7.8|/5 = 1.56. Así que X = 1.56 y L = 1.

2.15 Suponga que A y B tienen los siguientes valores: A = 2.5, B = 3.5. Encontrar los valores de A y B, que resultan después de cada lista de las proposiciones:

(*a*)
```
A = B
B = A
```

(*b*)
```
T = A
A = B
B = T
```

(*a*) La proposición A = B le dice al computador que borre el valor corriente de A y lo sustituya por el valor corriente de B; entonces,

$$A \leftarrow \cancel{2.5}\ 3.5$$

La siguiente proposición, B = A, le dice al computador que borre el valor que tiene B y lo sustituya por el valor que tiene A; entonces,

$$B \leftarrow \cancel{3.5}\ 3.5$$

Obsérvese que el valor de B no cambió.

(*b*) Las tres proposiciones conducen a:

$$T \leftarrow 2.5, \quad A \leftarrow \cancel{2.5}\ 3.5, \quad B \leftarrow \cancel{3.5}\ 2.5$$

Obsérvese que los valores de A y B han sido intercambiados al introducir una localización de almacenamiento auxiliar, para "guardar" el valor de una de las variables durante el intercambio.

2.16 Supóngase que las variables A, B y C han sido definidas anteriormente. Escribir una proposición Fortran que (*a*) doble el valor de A; (*b*) aumente el valor de B en 4; (*c*) disminuya el valor de C por el valor de A; (*d*) almacene en AVE el promedio de los valores de A, B y C, es decir, la suma de A, B y C dividida por 3; (*e*) almacene en DIST la raíz cuadrada de la suma de los cuadrados de A, B y C, o sea, $\sqrt{A^2 + B^2 + C^2}$.

(*a*) A = 2.0*A (*c*) C = C − A (*e*) DIST = SQRT(A**2 + B**2 + C**2)
(*b*) B = B + 4.0 (*d*) AVE = (A + B + C)/3.0

PROBLEMAS MISCELANEOS

2.17 Discuta las tres formas siguientes de escribir x^2 en Fortran:

(*a*) X*X (*b*) X**2 (*c*) X**2.0

En (*a*), x^2 se escribe en términos de una multiplicación y se evaluará así. En (*b*), X**2 se evaluará por multiplicación, como resultado de que X**2 se compila como X*X. En (*c*), el exponente es real, así que (*c*) se evalúa como EXP(2.0*ALOG(X)) y X debe ser positivo. (Ver la sección 2.8.) Claramente, (*a*) y (*b*) son preferibles a (*c*). En efecto, (*c*) puede dar un resultado ligeramente diferente en vista de los errores de redondeo introducidos al evaluar EXP y ALOG. Más aún, aunque (*a*) y (*b*) requieren el mismo tiempo de ejecución, (*a*) no requiere compilación extra.

2.18 Discutir los dos siguientes segmentos de programa:

(*a*)
```
XN = N
AVE = SUM/XN
```
(*b*)
```
AVE = SUM/FLOAT(N)
```

(Nótese que ambos se escriben evitando la mezcla de modos en la división.)

En (*a*), el entero N se convierte primero a real y se almacena en XN. Luego se computa el promedio como SUM/XN. Obsérvese que se requiere una localización adicional de almacenamiento. Por otro lado, (*b*) requiere solamente una proposición, puesto que usamos la función FLOAT en el cálculo. Luego (*b*) parece que tiene más ventajas que (*a*). Sin embargo, si el programa es muy largo y N necesita convertirse a real muchas veces, entonces sería más ventajoso convertir N a real de una vez por todas por XN = N, en lugar de usar FLOAT(N) a través de todo el programa.

Problemas suplementarios

CONSTANTES

2.19 Escribir las siguientes constantes como constantes enteras Fortran:

(*a*) 2,348 (*d*) 3.1214×10^3 (*g*) 23.51
(*b*) 5.31×10^3 (*e*) −37.0 (*h*) 1,250,000
(*c*) −531 (*f*) $21{,}500 \times 10^{-3}$ (*i*) $57{,}000.0 \times 10^{-2}$

2.20 ¿Por qué las siguientes constantes no son aceptables como constantes enteras Fortran?

(*a*) 2,578 (*b*) −784.0 (*c*) 37810000000 (*d*) 38E+2

2.21 Escribir las siguientes constantes como constantes reales Fortran y también en forma exponencial normalizada:

(*a*) 2,345 (*c*) 2,348,500 (*e*) .00005829
(*b*) −1.63 (*d*) -7.63×10^{-5} (*f*) 21×10^{18}

2.22 ¿Por qué las siguientes constantes no son aceptables como constantes reales Fortran?

(*a*) −3.6E134 (*c*) 52E−7 (*e*) E21
(*b*) 2,356.4 (*d*) 256 (*f*) −1,378.0

2.23 Determinar qué pares de constantes representan el mismo número:

(*a*) 43.6 4.36E01 (*d*) 12345 +12345
(*b*) 543 5.43E+02 (*e*) 0.234E6 234.E4
(*c*) 0.00004 4.0E−5 (*f*) 1.00 1.

VARIABLES

2.24 Considerar la siguiente lista de nombres. Determinar cuáles son aceptables como (1) variables enteras o (2) variables reales. Dar la razón por la cual las restantes no son aceptables como nombres de variables.

(*a*) MORE (*f*) A567B (*k*) AMOUNT
(*b*) LESS (*g*) RATE (*l*) A + 567
(*c*) NEITHER (*h*) X34.7 (*m*) A12345
(*d*) AAAAAA (*i*) GAMMA (*n*) IN-OUT
(*e*) 5PQR (*j*) KAPPA (*o*) OPERAND

OPERACIONES Y EXPRESIONES ARITMETICAS

2.25 Suponer que el computador no acepta mezcla de modos aritméticos. Determinar cuáles de las siguientes, son aceptables como expresiones Fortran, y encontrar sus valores. Identificar los errores en las expresiones inaceptables.

(*a*) 6.3 + 5.2E3 (*c*) 6.*4 (*e*) 0/3.4 (*g*) 9/−4
(*b*) −18/7 (*d*) 6.**4 (*f*) 6*−7 (*h*) −4/9

2.26 Suponer que el computador acepta mezcla de modos aritméticos. Encontrar el valor de cada expresión.

(*a*) 3 − 5*2.5 (*c*) 19/(2*5) (*e*) −3*2**3 (*g*) (7/3)/(6/5)
(*b*) 2.8 − 17/5 (*d*) 19/(2.*5) (*f*) −3*(2**3) (*h*) 2**2*2/3

2.27 Escribir la expresión Fortran correspondiente a cada expresión matemática.

(*a*) $(2x + y)(3z - 4w)$
(*b*) $4x^2y - 3xy + 7yz^3$
(*c*) $\left(\frac{a+b}{c+d}\right)^3$
(*d*) $\frac{x^5}{5!}$
(*e*) $\left(3 + \frac{a}{b}\right)^{m-1}$
(*f*) $\frac{\frac{a}{b} + 6}{x - \frac{y}{z}}$

2.28 Para cada una de las siguientes expresiones Fortran, encontrar una expresión equivalente, es decir, una expresión que no altere el orden de las operaciones al eliminar paréntesis superfluos.

(*a*) (X + Y) + Z (*c*) A + ((B*C)/D) (*e*) (X*(Y − Z))*(A**2)
(*b*) X + (Y + Z) (*d*) (A + (B**3))/(X*Y) (*f*) ((A**2) + (B**2)) − (D*(E/F))

2.29 Usando las funciones matemáticas de la tabla 2-2 (pág.), escribir la expresión Fortran correspondiente a cada expresión matemática.

(*a*) $\sqrt{5x^2 + 8y^2}$

(*b*) $\text{sen}(x - 2y) + e^{xy} - |x^2 - y^2|$

(*c*) $e^{|a|} - \dfrac{b^2}{|c|}$

(*d*) $\sqrt{|\cos(a - nb)|}$

(*e*) $\log_e(x + y)^2$

(*f*) $\log_{10}(a - b)^2$

(*g*) $\left|\sqrt{x - y} - \dfrac{z^3}{\cos(a + b)}\right|$

(*h*) $\sqrt{|\text{sen}(a - |b|)|}$

2.30 Las siguientes, son expresiones matemáticas con expresiones Fortran incorrectas. Escribir las expresiones Fortran correctas.

(*a*) $a + \dfrac{b}{c \cdot d}$ A + B/CD

(*b*) $\dfrac{x^{n+1}}{y^{n-1}}$ (X**N + 1)/(Y**N − 1)

(*c*) $\text{sen}(x + n)$ SIN(X + FLOAT(N)

(*d*) $\log_{10}|a \cdot b|$ LOG(ABS(AB)

PROPOSICIONES ARITMETICAS

2.31 ¿Cuáles de las siguientes, son aceptables como proposiciones Fortran?

(*a*) X = Y + Z
(*b*) X + Y = Z
(*c*) Y + Z = X
(*d*) XY = ZW
(*e*) XY = Z*W
(*f*) X*Y = Z*W
(*g*) ABC = DEF
(*h*) ABC = 2DEF
(*i*) A*B*C = D*E*F

2.32 Supongamos que X, Y, L y M contienen los siguientes valores: X = 3.1, Y = 4.6, L = 2 y M = −3. Encontrar los valores finales de A y J que resultan después de cada lista de proposiciones.

(*a*)
```
A = X - 2*Y + X/L**2
J = X - 2*Y + (X/L)**2
```

(*b*)
```
A = 8/M + 3*L
J = 8.0/M - Y
```

(*c*)
```
A = L/M*X/Y
J = L/(M + X)*Y
```

(*d*)
```
A = X + 2*Y
A = 2*A + 4
J = A
```

(*e*)
```
J = L + 3*M
J = J**2 + Y
A = X + J
```

(*f*)
```
J = SQRT(X*Y)
A = J*Y
J = A*M
A = A + ABS(FLOAT(J))
```

2.33 Suponer que X, Y y Z han sido definidos. Escribir una proposición aritmética que realice lo siguiente: (*a*) aumentar el valor de X en 3.2; (*b*) triplique el valor de Y; (*c*) elevar al cuadrado el valor de Z; (*d*) almacenar en PRDT el producto del valor de X, Y y Z; (*e*) almacenar en HYP la longitud de la hipotenusa de un triángulo rectángulo cuyos lados tienen longitud X e Y; (*f*) almacenar en AVE, el valor promedio de X, Y y Z.

2.34 Supongamos que A, B y C han sido definidos. Escribir un segmento de programa que intercambie los valores de A, B y C de tal forma, que A tome el valor de B, B tome el valor de C y C tome el valor de A. [Idea: Ver problema 2.15(*b*).]

Respuestas a problemas suplementarios seleccionados

2.19 (*a*) 2348 (*b*) 5310 (*c*) −531 (*d*) Imposible (*e*) −37 (*f*) Imposible
(*g*) Imposible (*h*) 1250000 (*i*) 570

2.20
(*a*) Contiene una coma.
(*b*) Contiene un punto decimal.
(*c*) Para la mayoría de los computadores, un entero no puede tener más de nueve caracteres.
(*d*) Un entero no puede ser escrito en forma exponencial

2.21
(*a*) 2345. ó 0.2345E4
(*b*) −1.63 ó −0.163E1
(*c*) 2348500. ó 0.23485E7
(*d*) −7.63E−5 ó −0.763E−4
(*e*) 0.00005829 ó 0.5829E−4
(*f*) 21.E18 ó 0.21E20

2.22
(*a*) Un exponente no puede tener más de dos dígitos en la mayoría de los computadores.
(*b*) Contiene una coma.
(*c*) No tiene punto decimal.
(*d*) No tiene punto decimal.
(*e*) Un exponente no puede aparecer solo (se puede escribir 0.0E21.)
(*f*) Contiene una coma.

2.23 (*a*) Sí (*b*) No—uno es entero. (*c*) Sí (*d*) Sí (*e*) No (*f*) Sí

2.24 (*a*) Entero. (*b*) Entero. (*c*) Tiene más de seis caracteres. (*d*) Real. (*e*) Empieza con un dígito. (*f*) Real. (*g*) Real. (*h*) El punto decimal no es permitido.
(*i*) Real. (*j*) Entero. (*K*) Real. (*l*) El signo más no es permitido. (*m*) Real
(*n*) El guión no es permitido. (*o*) Tiene más de seis caracteres.

2.25 (*a*) 5206.3 (*b*) −2 (*c*) Inaceptable (*d*) 1296. (*e*) Inaceptable
(*f*) Inaceptable puesto que * y − no pueden aparecer uno junto a otro. (*g*) −2 (*h*) 0

2.26 (*a*) −9.5 (*b*) −0.2 (*c*) 1 (*d*) 1.9 (*e*) −24 (*f*) −24 (*g*) 2 (*h*) 2

2.27
(*a*) (2.*X + Y)*(3.*Z − 4.*W)
(*b*) 4.*X**2*Y − 3.*X*Y + 7.*Y*Z**3
(*c*) ((A + B)/(C + D))**3
(*d*) X**5/(5.*4.*3.*2.*1.)
(*e*) (3. + A/B)**(M − 1)
(*f*) (A/B + 6.)/(X − Y/Z)

2.28
(*a*) X + Y + Z
(*b*) X + (Y + Z)
(*c*) A + B*C/D
(*d*) (A + B**3)/(X*Y)
(*e*) X*(Y − Z)*A**2
(*f*) A**2 + B**2 − D*(E/F)

2.29
(*a*) SQRT(5.*X**2 + 8.*Y**2)
(*b*) SIN(X − 2.*Y) + EXP(X*Y) − ABS(X**2 − Y**2)
(*c*) EXP(ABS(A)) − B**2/ABS(C)
(*d*) SQRT(ABS(COS(A − FLOAT(N)*B)))
(*e*) ALOG((X + Y)**2)
(*f*) ALOG10((A − B)**2)
(*g*) ABS(SQRT(X − Y**3) − Z**3/COS(A + B))
(*h*) SQRT(ABS(SIN(A − ABS(B))))

2.30 (*a*) A + B/(C*D)
(*b*) (X**(N + 1))/(Y**(N − 1)) ó X**(N + 1)/Y**(N − 1)
(*c*) SIN(X + FLOAT(N))
(*d*) ALOG10(ABS(A*B))

2.31 (*a*) Sí (*b*) No (*c*) No (*d*) Sí (*e*) Sí (*f*) No (*g*) Sí (*h*) No
(*i*) No

2.32 (*a*) A = −5.325, J = −3
(*b*) A = 4.0, J = −7
(*c*) A = 0.0, J = 92
(*d*) A = 28.6, J = 28
(*e*) A = 56.1, J = 53
(*f*) A = 54.8, J = −41

2.33
```
(a) X = X + 3.2
(b) Y = 3.*Y
(c) Z = Z**2
(d) PRDT = X*Y*Z
(e) HYP = SQRT(X**2 + Y**2)
(f) AVE = (X + Y + Z)/3.0
```

2.34
```
T = A
A = B
B = C
C = T
```

Capítulo 3

Entrada/salida numérica

3.1 INTRODUCCION

En este capítulo veremos cómo un computador lee datos externos (entrada) e imprime datos (salida). Esto es realizado por una proposición READ y WRITE, respectivamente. Estas proposiciones se llaman operaciones de entrada/salida o simplemente operaciones E/S. Como se estableció antes, suponemos que nuestro aparato de entrada es una lectora de tarjetas y nuestro aparato de salida es una impresora de líneas (a menos que otra cosa se establezca o se diga implícitamente).

Una proposición READ le dice al computador que lea información numérica o simbólica de un paquete de tarjetas (el paquete de datos) y una proposición WRITE le dice al computador que imprima información sobre la impresora. Normalmente, cada proposición WRITE o READ está acompañada por una proposición FORMAT. En entrada, la proposición FORMAT suministra al computador el tipo de información y su respectiva localización, en un registro de entrada. En salida, la proposición FORMAT también le dice al computador dónde se va a imprimir la información. Obviamente, las proposiciones FORMAT pueden ser muy detalladas; sin embargo, muchas instalaciones de computador permiten, los así llamados, utilidades de *entrada/salida sin formato*. Discutiremos primero estas E/S sin formato. (Algunos textos utilizan el término "Formato libre", en lugar del término "sin formato".)

3.2 ENTRADA SALIDA SIN FORMATO

Después de la lectura de esta sección, el lector puede proceder a la sección 3.10 y a los tres capítulos siguientes, antes de retornar al resto de éste (a menos que las facilidades de computador del lector no provean entrada/salida sin formato.)

La E/S sin formato discutida aquí puede variar de instalación a instalación con variaciones menores. Entonces, los lectores quedan advertidos para que consulten sus facilidades locales, para una información más exacta.

Una proposición típica PRINT sin formato es como sigue:

```
PRINT, A, B, C, AREA, M, N
```

Obsérvese que la palabra PRINT y cada variable, excepto la última, están seguidas por una coma. Esta proposición le dice al computador que imprima los valores de las variables A, B, C, AREA, M y N, en ese orden dado. Los valores serán impresos con un formato preasignado y los números reales serán *generalmente* impresos en forma exponencial. El número de valores que pueden imprimirse en una línea está también predeterminado. Sin embargo, el computador siempre empezará a imprimir una nueva línea, cuando encuentre una proposición PRINT.

Una proposición típica READ sin formato, es como sigue:

```
READ, A, B, C, LOT, AREA
```

Obsérvese que la palabra READ y cada variable distinta de la última, están seguidas de una coma. Esta proposición le dice al computador que "lea" cinco números, uno por cada variable, empezando desde la tarjeta que está encima en el paquete de datos. Si hay cinco o más números sobre la tarjeta, entonces los primeros cinco números se asignan a A, B, C, LOT, AREA respectivamente y la tarjeta es, entonces, descartada. Sin embargo, si hay menos de cinco números en la tarjeta, entonces el computador mantendrá la lectura del paquete de datos hasta que encuentre los cinco números. Esto es, la ejecución de la proposición READ no se finaliza hasta que estén todos los valores asignados a las variables listadas.

Nótese que una tarjeta de datos es descartada después de que se ha utilizado en lectura, luego una tarjeta que no se ha leido previamente estará siempre encima del paquete de datos.

Los números sobre el paquete de datos que se leen con proposiciones de entrada sin formato, deberán estar separados por comas (algunos centros de cómputo permiten que los números estén separados por espacios en blanco, en lugar de comas.) Los números reales pueden ser perforados en forma decimal o en forma exponencial. Obviamente, los números sobre la tarjeta y las variables correspondientes deberán ser del mismo tipo.

EJEMPLO 3.1

Supongamos que un computador ejecuta las dos proposiciones siguientes:

```
READ, I, J, K
READ, L, M
```

y supongamos que las tarjetas de datos están perforadas como sigue:

```
Primera tarjeta de datos:    222,76
Segunda tarjeta de datos:    38,175,55
Tercera tarjeta de datos:    194,58,567
```

Cuando el computador ejecuta la primera proposición READ, asigna 222 y 76 a I y J, respectivamente y luego va a la siguiente tarjeta y asigna 38 a K. Esto completa la ejecución de la primera proposición READ y la segunda tarjeta de datos es descartada, aun cuando los números 175 y 55 no fueron leidos. Cuando el computador ejecute la segunda proposición READ, leerá la tarjeta que está encima del paquete de datos. Entonces, el computador usa la tercera tarjeta de datos para asignar 194 a L y 58 a M. Luego, tendremos

$$I = 222,\ J = 76,\ K = 38,\ L = 194,\ M = 58$$

Algunas facilidades de computación requieren que los números sobre las tarjetas de datos estén separadas por espacios en blanco (denotados por b subindicado) en lugar de comas. En tales casos, las tarjetas de datos serán perforadas como sigue:

```
Primera tarjeta de datos:    222b76
Segunda tarjeta de datos:    38b175b55
Tercera tarjeta de datos:    194b58b567
```

3.3 INTRODUCCION A LA ENTRADA/SALIDA CON FORMATO

Las propiedades de entrada/salida sin formato (formato libre) fueron introducidas primero, para que los lectores puedan empezar a escribir programas y comunicarse con el computador tan pronto como sea posible. Esto obvia la complicación de los detalles de la escritura de las proposiciones FORMAT. Sin embargo, como una proposición de E/S sin formato se ejecuta con un formato preasignado, realmente no se tiene control de cómo se lee e imprime la información. Aquí discutimos específicamente, lo referente a la escritura de proposiciones E/S con formato. Limitaremos nuestra discusión a E/S con valores numéricos. Otros tipos de E/S serán tratados en capítulos posteriores.

En una instalación de computador grande podemos utilizar muchos aparatos diferentes para entrada y salida. Para distinguir tales aparatos se les han asignado números. La tabla 3-1 muestra números típicos para esas máquinas (esos números fueron utilizados primero en muchas instalaciones IBM y aún, hoy en día, se usan ampliamente.) En este texto supondremos que nuestro aparato de entrada es una lectora de tarjetas, unidad número 5, y nuestro aparato de salida es una impresora de líneas, unidad número 6.

Tabla 3-1

Aparato	Número
Cinta	1
Lectora de tarjetas	5
Impresora de líneas	6
Perforadora de tarjetas	7

Puesto que nuestro aparato de entrada es una lectora de tarjetas, nuestros datos de entrada serán dados al computador por medio de un paquete de tarjetas. Estas tarjetas son idénticas a las tarjetas sobre las cuales se perforan las proposiciones FORTRAN. Sin embargo, las reglas de perforación para las proposiciones FORTRAN no se aplican aquí, es decir, la totalidad de las 80 columnas de la tarjeta de datos puede ser utilizada y ninguna columna, por ejemplo la columna 6, tiene propósito especial alguno.

La entrada es perforada en ciertos bloques de columnas contiguas sobre la tarjeta de datos. La figura 3-1 muestra una tarjeta sobre la cual se han perforado cuatro números; el primer número perforado entre la columna 1 y 8 es 24587, el segundo número 193, es perforado entre las columnas 9 y 14, el tercer número, 8.725, es perforado entre las columnas 15 y 22 y el cuarto número, + 43450.75, es perforado entre las columnas 23 y 34. Tales bloques de columnas contiguas se llaman *campos*, y el número de columnas en un campo se llama su *longitud de campo*. Obsérvese que cada número es perforado *justificado a la derecha* en este campo; es decir, el último dígito del número aparece en la última columna de su campo. (Algunos textos usan el término ajustado a la derecha en lugar del término justificado a la derecha.)

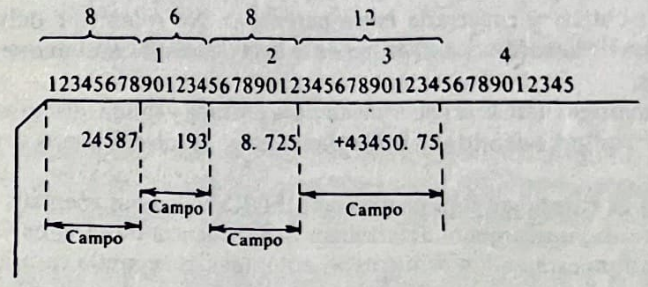


Fig. 3-1 Tarjeta de datos que muestra campos y longitudes de campo.

Supóngase que deseamos que los cuatro números anteriores sean asignados a las variables ID, LOT, RATE y PRICE, respectivamente. Esto se hace en Fortran por medio de una proposición READ y una proposición FORMAT que la acompaña y podría ser como sigue:

```
      READ(5, 100) ID, LOT, RATE, PRICE
100   FORMAT(I8, I6, F8.0, F12.0)
```

Consideremos los dos números entre paréntesis que siguen a la palabra READ. El primer número, 5, es el número del aparato de entrada que designa a una lectora de tarjetas. (Podríamos escribir 1 en lugar de 5 si deseamos utilizar una cinta en lugar de una lectora de tarjetas, por ejemplo, si el dato de entrada estuviera almacenado en una cinta magnética.) El segundo número, 100, especifica la proposición FORMAT que acompaña a esta proposición READ. (La proposición FORMAT provee la información necesaria de tipo y de localización para las variables.) La escogencia de 100 es completamente arbitraria. Siguen los nombres de las variables y éstas son listadas en el orden en que sus valores fueron perforados sobre la tarjeta de datos.

Este orden también se aplica a las cuatro entradas en la proposición FORMAT, I8, I6, F8.0 y F12.0. Estas entradas son llamadas *especificaciones de campo* (algunas veces *descriptores de campo*) ó *códigos de formato*. Cada entrada contiene una letra que denota el tipo de dato, por ejemplo entero o real, y un número que denota la longitud del campo, y por consiguiente, la localización de campo del dato correspondiente. En nuestro caso, la primera especificación de campo I8 indica que el primer número es un entero y que puede ser encontrado entre las primeras ocho columnas de la tarjeta. La segunda especificación, I6, indica que el segundo número es un entero y que puede ser encontrado entre las seis columnas siguientes de la tarjeta. La tercera especificación, F8.0, indica que el tercer número es real y que puede ser encontrado entre las ocho columnas siguientes de la tarjeta y la cuarta especificación, F12.0, indica que el cuarto número es real y que puede encontrarse dentro de las doce columnas siguientes de la tarjeta.

En este ejemplo, la especificación de campo de un entero es de la forma *Iw*, y la especificación de campo de un número real es de la forma *Fw*.0, donde *w* denota la longitud de campo. También, hemos convenido que los números reales son perforados en forma decimal.

3.4 ESPECIFICACIONES DE CAMPO PARA ENTRADA Y ENTRADA CON FORMATO

El computador lee datos (entrada) por medio de una proposición READ y una proposición FORMAT que la acompaña, como se ilustró en la sección anterior. La forma general de tales proposiciones es como sigue:

```
    READ(m, n) lista de variables
n   FORMAT(lista de formatos)
```

Específicamente, una proposición READ empieza con la palabra READ. Esta es seguida por un entero m, que especifica el número del aparato de entrada y un número de proposición n, que especifica la proposición FORMAT; m y n están separados por una coma y encerrados entre paréntesis. La lista de variables que sigue, contiene los nombres de las localizaciones de memoria, separadas por comas, en donde se almacenarán los datos de entrada.

La proposición FORMAT acompañante, que debe ser rotulada por el número de proposición n, empieza con la palabra FORMAT. Esta es seguida por una lista de proposiciones de campo (códigos de formato) separados por comas y encerrada entre paréntesis. No solamente deberá estar el orden de las especificaciones de campo de acuerdo con el orden de la lista de variables, sino que deberán estar de acuerdo en los tipos de variables.

Aunque las proposiciones FORMAT se utilizan para entrada y salida, discutiremos ahora el significado de las especificaciones de campo, cuando son utilizadas en entrada (discutiremos su significado en salida en la sección 3.6.)

Las especificaciones de campo en una proposición FORMAT, contienen sus respectivas longitudes de campo w_1, w_2,... En entrada, únicamente determinan una secuencia de campos sobre una tarjeta de datos como sigue: el primer campo está en las primeras w_1 columnas, el segundo campo está en las w_2 columnas siguientes y así, sucesivamente. Las longitudes de campo en el ejemplo de la sección precedente eran 8, 6, 8 y 12 respectivamente.

Nótese que cuando se lee con especificaciones de campo *numéricas* (que serán discutidas posteriormente), se aplican las dos reglas siguientes:

1. Los espacios en blanco son interpretados como ceros.
2. Valores sin signo son tomados como positivos.

Se hace énfasis en el hecho de que la regla 1 no se mantiene si se utilizan especifaciones de campo no numéricas.

Campo I

La especificación de campo general para lectura de un entero es

Iw

donde el carácter *I* indica el tipo entero y w es un constante entero sin signo, que da la longitud de campo.

Puesto que, los blancos en campos numéricos se toman como ceros, es imperativo que un entero sea perforado

justificado a la derecha

en este campo, de tal forma que el último dígito del entero aparezca en la última columna del campo dado.

Consideremos las proposiciones

```
    READ(5, 20) M, N
20  FORMAT(I3, I5)
```

Estas proposiciones le dicen al computador que lea dos enteros de la tarjeta de datos y que los almacene en M y N respectivamente. La especificación para M es I3 y para N es I5. Puesto que, las longitudes de campo son 3 y 5 respectivamente, el entero que se encuentra en las columnas 1-3, es asignado a M y el entero que se encuentra en las columnas 4-8, es asignado a N.

Supongamos que los enteros 40 y −123 sean los valores para M y N, respectivamente. Entonces, la tarjeta de datos debe ser perforada como la tarjeta A en la figura 3-2. De otra forma, si la tarjeta de datos fuera

perforada como la tarjeta B en la figura 3-2, entonces la proposición READ anterior asignaría un valor de 400 a M y de −1230 a N, puesto que los espacios blancos en las columnas 3 y 8 se interpretan como ceros.

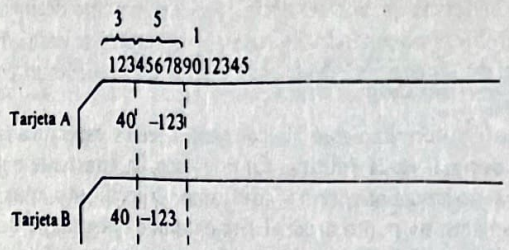


Fig. 3-2

Campo F

Las constantes reales en forma decimal pueden leerse usando campos F. La forma general es

Fw.d

La *F* indica que el tipo de dato es real y que el número está escrito en forma decimal, *w* es una constante entera sin signo que denota la longitud del campo y *d* es un entero sin signo que representa el número de dígitos decimales. Sin embargo, *d es ignorado en entrada*, si el número perforado en el campo *tiene un punto decimal.*

Supongamos que se quiere leer el número −403.125 para asignarlo a la localización A utilizando

```
      READ(5, 20) A
20    FORMAT(F12.4)
```

Puesto que la longitud del campo es 12, se asignará a A el número que aparezca en las primeras doce columnas de la tarjeta de datos. Entonces, una cualquiera de las tres primeras tarjetas en la figura 3-3 puede utilizarse como la tarjeta de datos; puesto que el número está en forma decimal.

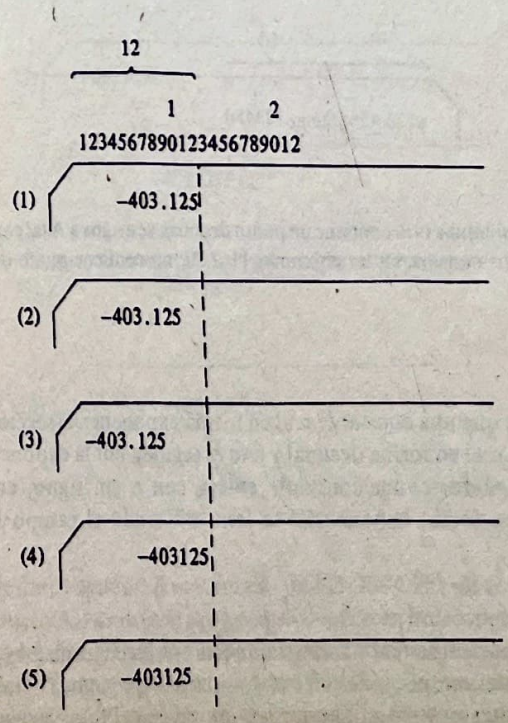


Fig. 3-3

El valor de A no tuvo que ser perforado justificado a la derecha en su campo, como era el caso para las constantes enteras. Aunque los espacios en blanco se interpretan como ceros en los campos numéricos, añadir ceros al comienzo o al final de un número en forma decimal no cambia su valor; por ejemplo, -403.125 y -403.12500 tienen el mismo valor. Entonces, como se estableció anteriormente, cuando se lee con campo F, si el número es perforado en forma decimal (es decir, contiene un punto decimal) y yace dentro del campo especificado, el entero d en $Fw.d$ no se tendrá en cuenta, y el número se leerá como aparece en la tarjeta de datos. Se hace notar que el número de cifras significativas que retiene un computador, es una norma dependiente de la máquina.

Muchos compiladores no necesitan que las constantes reales especificadas por los campos F sean perforadas con un punto decimal en la entrada. En este caso, la constante entera d en $Fw.d$ cumple su objetivo. Específicamente, si no hay punto decimal en el campo especificado, entonces el número leido tendrá exactamente d dígitos decimales; un punto decimal será encajado entre las columnas d y $(d + 1)$ contando hacia la izquierda desde la última cifra significativa a la derecha del campo. Por ejemplo, supongamos que el par de proposiciones anteriores READ-FORMAT se utilizan para leer la tarjeta (4) en la figura 3-3. Como no hay punto decimal entre las columnas 1 y 12, cuando se lee con la especificación de campo F12.4, un punto decimal será asumido entre la 4 y 5 columnas desde el lado derecho del campo, es decir, entre las columnas 9 y 8. En consecuencia, el número asignado a A es -40.3125. Análogamente, si el mismo par de proposiciones READ-DORMAT se utiliza para leer la tarjeta (5) en la figura 3-3, entonces el número asignado a A es -4031.25

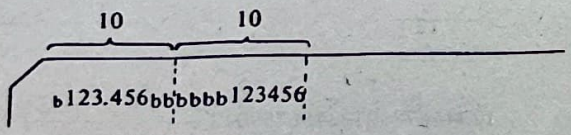
La norma anterior elimina la necesidad de perforar el punto decimal en constantes reales cuando se prepara el paquete de datos (esto claramente será una ventaja en tiempo para aplicaciones comerciales.) Sin embargo, cuando se utiliza esta norma, la posición del número perforado afecta el número asignado. En consecuencia, los programadores principiantes quedan advertidos cuando sigan esta norma, es decir, perforar el punto decimal cuando usen el campo F.

EJEMPLO 3.2

Supóngase que el par de proposiciones

```
      READ(5, 21) A, B
21    FORMAT(F10.2, F10.2)
```

se usa para leer una tarjeta perforada como sigue:



Puesto que el número en las columnas 1-10 contiene un punto decimal, se asigna a A tal como aparece; luego A contendrá el valor de 123.456. El segundo número, en las columnas 11 a 20, no contiene punto decimal, luego será leido como 1234.56 y asignado a B.

Campo E

Recalquemos, primero, que una constante real en forma exponencial (sección 2.2) consta de dos partes: la primera, es una constante real en forma decimal y ésta es seguida por la exponencial (la segunda parte) que empieza con el carácter E, seguido por una constante entera con o sin signo, con a lo sumo, dos dígitos.

Las constantes reales en forma exponencial se leen utilizando el campo E. La forma general de la especificación de campo es

Ew.d

La letra *E* indica que el dato es de tipo real y la constante está escrita en forma exponencial, *w* es un entero sin signo que denota la longitud del campo, y *d* es un entero sin signo que indica el número de dígitos decimales. En la misma forma que en el campo F, se ignora *d* en entrada, si el número está perforado con un punto decimal.

Supongamos que el número −4.03125E+02 se lee para asignarlo a la localización A, utilizando las proposiciones

```
      READ(5, 30) A
30    FORMAT(E18.5)
```

Obsérvese que la longitud del campo es de 18; así que a A le será asignado el número que aparezca en las primeras 18 columnas de la tarjeta de datos. Luego, cualquiera de las cinco primeras tarjetas de la figura 3-4 se

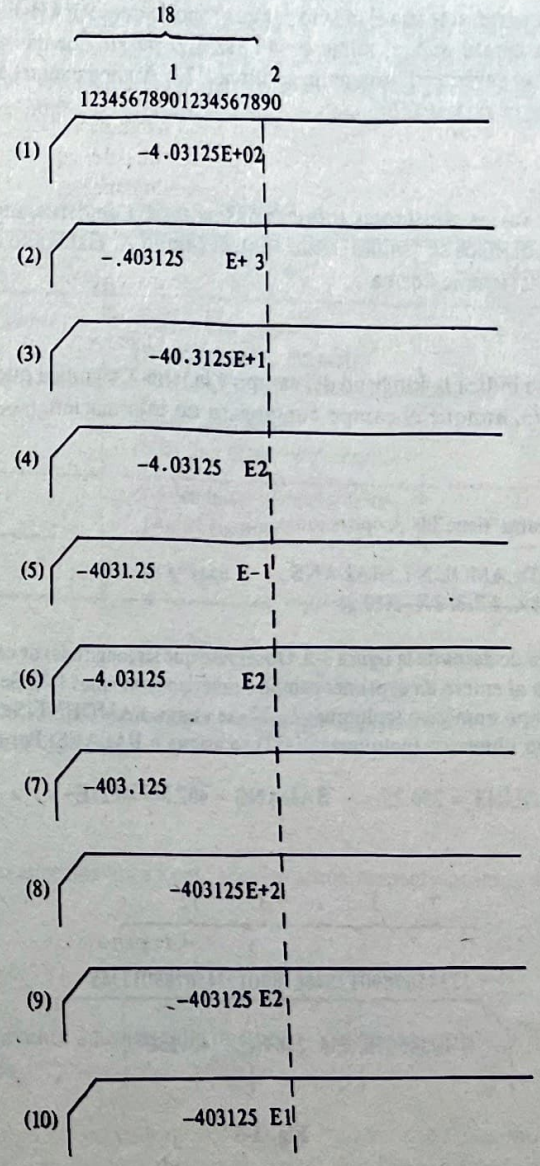


Fig. 3-4

puede usar como la tarjeta de datos dando a A el valor de -0.403125×10^3. Si se usa la sexta tarjeta, entonces −4.03125E + 20 será asignado a A, puesto que el espacio en blanco en la columna 18 se interpreta como cero. En consecuencia, en el campo E es esencial que la parte exponencial sea justificada a la derecha del campo.

Si una constante real tiene un exponente cero (es decir, E + 00), entonces, no se necesita perforar la parte exponencial. En otras palabras, si el número no es perforado con una parte exponencial, entonces la especificación de campo *Ew.d* se debe tratar como si fuera *Fw.d*. Por esta razón, la séptima tarjeta en la

figura 3-4 también se puede utilizar para asignar el número −4.03125E+02 a A. (En vista de esta flexibilidad, algunos programadores prefieren usar la forma E en entrada, sin tener en cuenta si la tarjeta de datos está perforada con o sin exponente.)

Como en el campo F, el uso de un punto decimal en entrada es opcional cuando se usa el campo E. Si no hay un punto decimal en el campo especificado (a la izquierda de E), entonces se presume en existencia entre las columnas d y $d+1$, contando hacia la izquierda, desde la primera columna que está a la izquierda del carácter E. Por ejemplo, supongamos que las proposiciones READ-FORMAT anteriores se usan para leer la tarjeta (8) en la figura 3-4. Puesto que no hay punto decimal entre las columnas 1 y 18, el entero 5 en la especificación de campo E18.5 hará que se encaje un punto decimal entre la quinta y la sexta columna a la izquierda del carácter A, esto es, entre las columnas 10 y 11; por lo tanto, el número asignado a A es aún −4.03125E+2. Por otra parte, si se usa el mismo par de proposiciones READ-FORMAT para leer la tarjeta (9) de la figura 3-4, se asignará a A el número −40.3125E2, puesto que hay un punto decimal entre las columnas 11 y 12. (Aquí el carácter E está en la columna 17.) Análogamente, el número −50.3125E1 será asignado a A, si se utiliza la tarjeta (10).

Campo X

Se pueden distinguir varias constantes sobre la misma tarjeta de datos, separándolas con uno o más espacios en blanco. Estos blancos se "saltan" utilizando el campo X. Hablando en términos generales, una especificación de campo X tiene la forma

$$wX$$

w es un entero sin signo que indica la longitud del campo y la letra X significa que el campo correspondiente de w columnas será *saltado*, aunque el campo contenga o no información.

EJEMPLO 3.3

Supóngase que un programa tiene las proposiciones

```
      READ(5, 75) ID, AMOUNT, BALANS
75    FORMAT(I7, 3X, F7.2, 3X, E12.2)
```

y supóngase que se lee la tarjeta de datos de la figura 3-5. Obsérvese que las longitudes de campo son 7, 3, 7, 3 y 12, respectivamente. Así, ID es asignado al entero en el primer campo numérico (columnas 1-7). Se saltan las columnas 8 y 10. El número real en el segundo campo numérico (columnas 11-17- se asigna a AMOUNT. Se saltan las columnas 18-20. El número real en el tercer campo numérico (columnas 21-32) se asigna a BALANS. Por consiguiente,

ID = 12345, AMOUNT = 250.25, BALANS = 402.5 = 40.25E+1

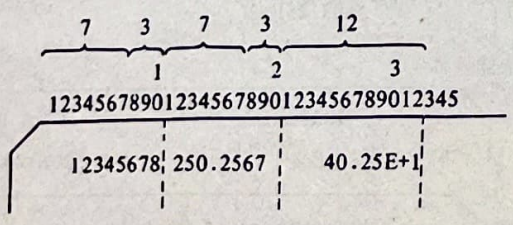


Fig. 3-5

3.5 PROPOSICIONES WRITE CON FORMATO Y CONTROL DE CARRO

El computador está instruido para sacar datos por medio de una proposición WRITE y una proposición FORMAT que la acompaña y que tienen la siguiente forma:

```
      WRITE(m, n) lista de variables
n     FORMAT (lista de formatos)
```

Específicamente, la proposición WRITE empieza con la palabra WRITE; ésta va seguida por un entero, m, que especifica el número del aparato de salida y un número de proposición n que indica la proposición FORMAT que lo acompaña; m y n están separados por una coma y encerrados entre paréntesis. Siguen las variables, cuyos valores serán impresos, separadas por comas. Supondremos que se usa una línea de impresora para escribir nuestra salida, así que en todos los ejemplos, el número del aparato de salida será $m = 6$. (Ver tabla 3-1.)

La proposición FORMAT acompañante debe estar rotulada por el número de proposición n, y tendrá la misma forma que en entrada, es decir, empieza con la palabra FORMAT y sigue con una lista de especificaciones de campo que están separadas por coma y encerradas entre paréntesis. El significado de las especificaciones de campo en salida será explicado en la sección siguiente.

Cada par WRITE-FORMAT generará una cadena de salida (registro) de caracteres. (Un espacio en blanco es un carácter y algunas veces será denotado por "b", esto es, por un subíndice b.) El primer carácter en una cadena de salida, controla el movimiento del carro de la impresora de líneas, de acuerdo a las instrucciones de la tabla 3-2, y no se imprime. Específicamente, el computador examina el primer caracter de la

Tabla 3-2

Primer carácter	Instrucción	Formato de entrada
b (blanco)	Avance una línea (espacio sencillo normal)	1X ó ' '
0 (cero)	Avance dos líneas (salte una línea, doble espacio)	'0'
1	Avance hasta el inicio de la página siguiente	'1'
+	No avance	+

cadena de salida da la instrucción de control de carro y luego imprime los caracteres restantes de la cadena de salida. Por ejemplo, la cadena de salida

$$_{bbbbb}12345_{bb}67.89$$

le dice al computador que avance una línea y entonces imprima respectivamente 4 blancos, 12345, 2 blancos y 67.89; la cadena de salida.

$$0_{bbbb}12345_{bb}67.89$$

le dice al computador que avance 2 líneas (salte una línea) y entonces imprima 4 blancos, 12345, 2 blancos y 67.89; y la cadena de salida

$$1_{bbbb}12345_{bb}67.89$$

le dice al computador que avance a una nueva página y luego imprima 4 blancos, 12345, 2 blancos y 67.89.

La proposición FORMAT en un par WRITE-FORMAT le dice al computador la clase de tipos que tienen los datos de salida y donde aparecen esos datos en la cadena de salida. Específicamente, cada valor de la lista de variables en la proposición WRITE es asignado a un bloque de espacios (llamados *campos*) en la entrada correspondiente en la proposición FORMAT. El valor numérico de las variables es siempre colocado

justificado a la derecha

en su campo; es decir, de tal forma que el número termine en el último espacio del campo. Ilustraremos con un ejemplo, aunque las entradas de los formatos serán discutidas en detalle en la sección siguiente.

EJEMPLO 3.4

Supóngase que deseamos imprimir los valores de J, K y L, en ese orden, donde J = 128, K = 72 y L = −179. Esto puede hacerse con el siguiente par de proposiciones:

```
      WRITE(6, 20) J, K, L
20    FORMAT(I8, I6, I8)
```

El 6 en la proposición WRITE le dice al computador que el aparato de salida es una impresora de líneas; el 20 le dice que los valores serán impresos de acuerdo a la proposición FORMAT rotulada con 20. (El número 20 es escogido arbitrariamente.)

Los primeros tres campos en la cadena de salida determinada por el par WRITE-FORMAT contienen los siguientes caracteres:

bbbbb128	(I8)	El valor entero de J es justificado a la derecha en el primer campo de longitud 8; es decir, el valor de J se completa a la izquierda con 5 caracteres en blanco.
bbbb72	(I6)	El valor de K es generado justificado a la derecha, en el siguiente campo de longitud 6.
bbbb−179	(I8)	El valor entero de L es generado justificado a la derecha, en el siguiente campo de longitud 8.

En consecuencia, la cadena de salida contiene los siguientes caracteres:

Cadena de salida: bbbbb128bbbb72bbbb−179

Por lo demás, la cadena de salida le dice al computador que avance una línea, y luego imprima 4 blancos, 128, 4 blancos, 72, 4 blancos y −179. La salida impresa aparece en la figura 3-6. Las páginas para impresión son de varios tamaños, se acepta que la que aparece en la figura 3-6 tiene 132 columnas.

Resumiendo, el computador realiza los siguientes pasos en una instrucción WRITE-FORMAT:

1. Genera la cadena de salida dada de acuerdo con el par de instrucciones WRITE-FORMAT.
2. Examina el primer carácter y realiza la instrucción de control de carro.
3. Imprime los caracteres restantes de la cadena de salida.

Hacemos notar que después de completarse la instrucción WRITE, la impresora de líneas siempre pasa sobre la línea que imprimió. Entonces, el primer carácter generado por cualquier proposición de salida sucedánea tendrá que ser utilizado de nuevo para control de carro.

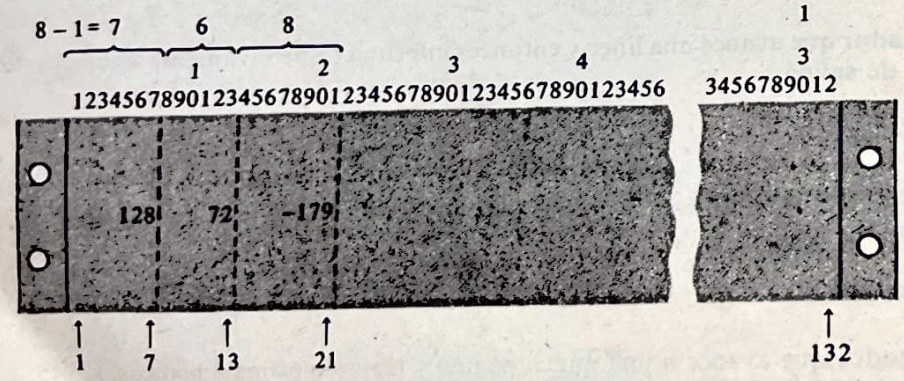


Fig. 3-6

Nota: Como un buen hábito de programación, generalmente se da una señal explícita de control de carro. Esto se hace escogiendo una de las entradas de la tabla 3-2 como la primera entrada de una proposición FORMAT. Esto se discutirá y explicará en la sección siguiente. Tal hábito elimina errores como los ilustrados en los problemas 3.12(*b*), 3.32(*f*) y 3.33(*f*). Obsérvese que la señal explícita de control de carro no fue dada en el ejemplo anterior, pero la señal de control de carro fue incorporada en la especificación de campo I8 en el que se dio un campo mayor que el valor de J.

3.6 ESPECIFICACIONES DE CAMPOS DE SALIDA

Como se discutió previamente, la proposición FORMAT que acompaña a una proposición WRITE especifica los tipos de los datos y las localizaciones de la información que se va a imprimir. En esta sección, cubriremos los mismos tipos de especificaciones de campo que fueron analizados en la entrada:

wX, *Iw*, *Fw.d*, *Ew.d*

Hacemos énfasis en el hecho de que el significado de estas especificaciones en salida pueden diferir del de entrada. De una parte, *todas las constantes numéricas son impresas justificadas a la derecha de sus campos.* Además, la *d* de *Fw.d* siempre juega un papel mucho más importante en salida que en entrada, puesto que algunas veces *d* puede ser eliminado en entrada.

Campo X

Se pueden separar constantes sobre la misma línea por medio de espacios en blanco, utilizando el campo X. La forma general de la especificación de campo es

wX

donde *w* denota la longitud del campo. El efecto de *wX* es *generar w espacios en blanco en la cadena de salida.*

En particular, si la primera entrada de la proposición FORMAT es 1X, entonces se genera un blanco como primer carácter de la cadena de salida. Este blanco, utilizado para control de carro, avanza el carro una línea. En otras palabras, las proposiciones FORMAT de la forma

n FORMAT(1X,...)

siempre conduce a espaciamiento simple normal. Análogamente,

n FORMAT(10X,...)

significará siempre espaciamiento simple con 9 columnas en blanco

Campo I

Las constantes enteras se imprimen usando el campo I. La forma general de la especificación de campo es

Iw

donde *I* denota el tipo entero y *w* la longitud del campo.

Por ejemplo, supongamos que M contiene 250 y N contiene 46 y se ejecuta

```
      WRITE(6, 10) M, N
10    FORMAT(1X, I6, 3X, I4)
```

la entrada 1X genera un blanco: ␣. La entrada I6 que corresponde a M genera ␣␣␣250; esto es, el valor de M se genera justificado a la derecha en el campo siguiente de longitud 6. La entrada 3X genera tres blancos: ␣␣␣. La entrada I4 que corresponde a N genera ␣␣46; esto es, el valor de N es generado justificado a la derecha en el siguiente campo de longitud 4. Entonces, la cadena de salida generada es:

Cadena de salida: ␣␣␣␣250␣␣␣␣␣46

Puesto que el primer carácter de la cadena de salida es un blanco, que significa espaciamiento simple, la impresora avanzará una línea e imprimirá el resto de caracteres:

Caracteres impresores: ␣␣␣250␣␣␣␣␣46

Si la longitud que se deja es insuficiente, es decir, si un valor numérico tiene más caracteres que los especificados en la longitud del campo, algunos compiladores truncarán el valor sobre la izquierda (o la derecha) e imprimirán el resto de caracteres. En el ejemplo anterior, si N contiene 54321, entonces se imprime

␣␣␣250␣␣␣4321

y no se da ningún mensaje de diagnóstico de error. Afortunadamente, la mayoría de los compiladores bien construidos señalarán insuficiencia de longitud de campo imprimiendo asteriscos en el campo designado. Utilizando el ejemplo anterior, de nuevo con N = 54321, se imprimirá

$_{bbb}250_{bbb}****$

Los programadores deberán cuidar de que la longitud de campo sea lo suficientemente grande como para acomodar todos los valores posibles de la variable.

Campo F

Se pueden imprimir las constantes reales en forma decimal, utilizando el campo F. La forma general de la especificación de campo es

Fw.d

donde *F* denota el tipo del dato y *w* la longitud del campo. El entero *d* denota el *número de dígitos decimales* y está siempre acompañado de redondeo. En otras palabras, *Fw.d* le dice al computador que redondee el número real correspondiente a *d* cifras decimales e imprima el número justificado a la derecha de su campo de longitud *w*.

EJEMPLO 3.5

Supongamos que ID = 125 y AMOUNT = 450.2462 y que se ejecuta

```
      WRITE(6, 27) ID, AMOUNT
27    FORMAT(I8, 3X, F10.2)
```

Obsérvese que las especificaciones de campo contienen las longitudes 8, 3 y 10, respectivamente. La especificación de campo I8 que corresponde a ID, le dice al computador que genere el valor de ID, justificado a la derecha en el primer campo de longitud 8, en la cadena de salida, así que se genera $_{bbbbb}125$ · 3X genera tres espacios en blanco: $_{bbb}$ · F10.2 le dice al computador que redondee el valor de AMOUNT a 2 decimales, es decir, 450.25 y que genere ese valor justificado a la derecha del siguiente campo de longitud 10, luego se genera $_{bbbb}450.25$. Entonces la cadena de salida generada es:

Cadena de salida: $_{bbbbb}125_{bbbbbbb}450.25$

Puesto que el primer carácter se utiliza para control de carro, la salida aparecerá como la página impresa en la figura 3-7.

Hacemos hincapié en el hecho de que el número 450.25 ocupa 6 lugares en la salida, puesto que el punto decimal utiliza un espacio.

Fig. 3-7

En la misma forma que con el campo I, si la longitud es insuficiente, es decir, si el valor numérico en el campo F contiene más caracteres que la longitud del campo, entonces el número puede ser truncado sobre su izquierda (o derecha) o se pueden imprimir asteriscos. Luego, cuando usamos el campo F, se debe tener cuidado para que la longitud del campo sea lo suficientemente grande como para que acomode el número. Específicamente, si *d* es el número de decimales deseado y *n* es el número de enteros posibles del número, entonces *w* deberá al menos satisfacer que $w \geq n + d + 2$. El 2 da los espacios para el signo y para el punto decimal. Por ejemplo, si $|X| < 1000$ y necesitamos solamente dos dígitos decimales, entonces F7.2 es suficiente: $w = 3 + 2 + 2$, donde los tres espacios se usan para la parte entera de X.

Se debe tener cuidado también de que el número de dígitos decimales sea suficientemente grande. Por ejemplo, supongamos que A contiene el número 0.246E−05, y la especificación de campo dada es F12.4. (Los números reales son almacenados en un computador en forma científica, ver apéndice A.) Entonces, sólo se imprime

0.0000

puesto que el valor de A es 0.00000246 en forma decimal, cuando se redondea a 4 cifras decimales será 0.0000.

Campo E

Se pueden imprimir constantes reales en forma exponencial usando el campo E. La forma general de la especificación de campo es

Ew.d

donde *E* denota el tipo de dato y *w* la longitud de campo. El entero *d* denota el *número de dígitos significativos* y está acompañado de un redondeo. Más aún, en salida, las constantes reales se imprimen en *forma exponencial normalizada*, es decir, como constantes entre 0.1 y 1.0 (ó −0.1 y −1.0) junto con el exponente apropiado que siempre contiene 4 caracteres, E ± XX. En otras palabras, *ed.d* le dice al computador que redondee el número real correspondiente a *d* cifras significativas y que lo imprima en forma exponencial normalizada justificado a la derecha de su campo de longitud *w*. (Notemos que en forma exponencial normalizada, el número de cifras significativas es también igual al número de dígitos decimales.)

EJEMPLO 3.6

Supóngase que A = 251.381 y B = 0.0081626 y que se ejecuta

```
      WRITE(6, 22) A, B
 22   FORMAT(E15.4, E12.3)
```

Como los números reales (en forma E ó F) son almacenados internamente en forma exponencial normalizada (ver apéndice A), A contiene +0.251381E+03, y B contiene +0.81626E−02. La entrada E15.4 le dice al computador que redondee el valor de A a cuatro cifras significativas, esto es, a 0.2514E + 03 y que lo coloque justificado a la derecha en los primeros quince espacios de la cadena de salida. La entrada E12.3 le dice al computador que redondee el valor de B a 3 cifras significativas, esto es, a 0.816E−02 y que lo coloque justificado a la derecha en los 12 espacios siguientes de la cadena de salida. En consecuencia, ellos aparecerán como se muestra en la figura 3-8.

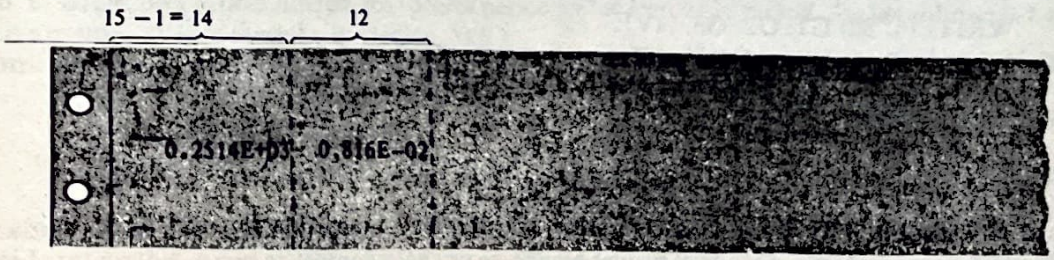


Fig. 3-8

Para dejar espacio suficiente para imprimir números reales en forma E, calculamos *w* como sigue: si se desean *d* dígitos de cifras signficativas, entonces la longitud de campo debe satisfacer que

$$w \geq d + 7$$

Esto se obtiene como sigue: se necesitan 4 espacios para el exponente E ± XX, un espacio para el signo, uno para colocar el cero y uno para el punto decimal. La figura 3-9 ilustra el cálculo.

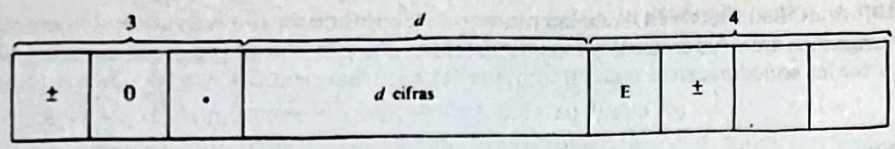


Fig. 3-9

Concluimos las secciones de entrada/salida con los siguientes comentarios:

1. Una proposición FORMAT puede contener muchas especificaciones de campo. El hecho importante es que los campos sean contiguos, es decir, uno junto a otro.
2. Para leer o imprimir valores, el orden en el que se listan las variables será correspondiente uno a uno con el orden de la especificación de campo numérico (o carácter numérico) en la proposición FORMAT correspondiente. También deben estar de acuerdo en los tipos.
3. Cada proposición READ o WRITE debe tener una proposición FORMAT acompañante. Sin embargo, una proposición FORMAT puede acompañar (puede ser referida por) varias proposiciones E/S. Además, las proposiciones FORMAT no son ejecutables.

3.7 CAMPO LITERAL

Supongamos que deseamos imprimir el encabezamiento: POBLACION DE USA. Esto se puede realizar *encerrando el mensaje entre apóstrofes*, dentro de la proposición FORMAT, como sigue:

```
      WRITE(6, 30)
30    FORMAT(1X, 'POBLACION DE USA')
```

Cuando se ejecuta la proposición WRITE anterior, todos los caracteres dentro de los *apóstrofes* serán generados. La cadena de salida contiene:

Cadena de salida: $_{b}$POBLACION$_{b}$DE$_{b}$USA

donde el primer blanco es generado por 1X. Teniendo en cuenta el control de carro, se imprime el mensaje POBLACION DE USA.

Se pueden imprimir mensajes de caracteres junto con valores numéricos. Supongamos que G1 = 84.5, G2 = 78.4, G3 = 94.2 y AVE = 85.7. Entonces las instrucciones

```
      WRITE(6, 80) G1, G2, G3, AVE
80    FORMAT(6X, 'GRADOS', 3F8.1, 4X, 'PROMEDI', F10.1)
```

generan la siguiente cadena de salida:

Cadena de salida: $_{bbbbbb}$GRADOS$_{bbbb}$84.5$_{bbbb}$78.4$_{bbbb}$94.2$_{bbbb}$PROMEDI$_{bbbbbb}$85.7

La salida aparecerá como en la hoja impresa en la figura 3-10. Obsérvese que la entrada 6X causará solamente $5 = 6 - 1$ blancos para imprimir, porque el primer blanco, se usa para control de carro. Nótese también que

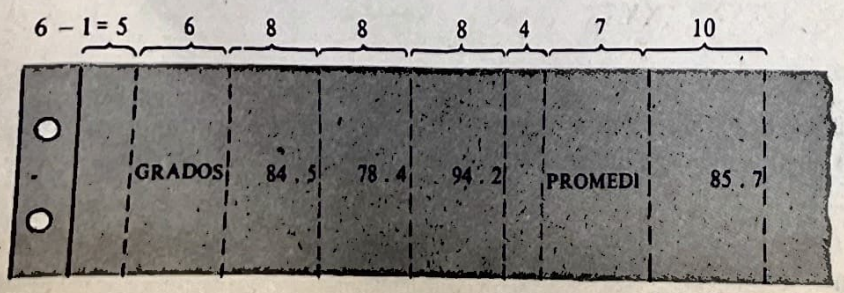


Fig. 3-10

los mensajes GRADOS y PROMEDI ocupan campos de longitud 6 y 7 respectivamente, o sea el número de caracteres del mensaje. (Notemos que 3F8.1 es una abreviación de F8.1, F8.1, F8.1. Esto se cubre en la sección 3.9.)

Se utiliza bastante el campo literal para control de carro. Supongamos que deseamos imprimir el mensaje POBLACION DE USA al comienzo de una nueva página. Esto puede realizarse con las proposiciones

```
      WRITE(6, 20)
20    FORMAT('1', 'POBLACION DE USA')
```

La primera entrada '1', es la instrucción de control de carro, para que vaya a una nueva página. De igual manera, usaríamos '0' como el primer formato de entrada, si deseáramos saltar una línea antes de imprimir POBLACION DE USA, o usaríamos '+' como el primer formato de entrada, si deseáramos que el computador imprimiera sobre la misma línea. Esas entradas están listadas en sus posiciones correspondientes en la tercera columna de la tabla 3-2 sobre instrucciones de control de carro.

3.8 REGISTROS, REGISTROS MULTIPLES, DIAGONAL

Un *registro* es un grupo de datos que corresponde a variables en una lista de E/S. La máxima longitud de un registro está limitada por el aparato particular de E/S que se utilice: 80 caracteres para una tarjeta, 72 caracteres para máquina de escribir por consola y 132 caracteres para impresoras de línea. Naturalmente, la longitud real del registro dado por una instrucción de E/S es proporcionada por la proposición FORMAT.

Concluyendo, cada una de nuestras instrucciones de E/S ha transmitido un registro de información: cada par READ-FORMAT lee de una tarjeta de entrada y cada par WRITE-FORMAT escribe una línea de salida. Sin embargo, es posible utilizar una sola proposición FORMAT para definir registros múltiples. A continuación vamos a estudiar estas dos formas.

Caso A

Recalquemos que la ejecución de una proposición READ/WRITE no se completa hasta que todas las variables sean asignadas/impresas. Supóngase que se ejecute

```
      READ(5, 20) A, B, C, D
20    FORMAT(F8.2, 2X, F15.7)
```

Nótese que hay más variables (4) que especificaciones numéricas (2). Primero las variables A y B se leen de la tarjeta 1 en las columnas 1-8 y 11-25 con especificaciones F8.2 y E15.7, respectivamente. Puesto que hay más de dos variables para leer, se usa de nuevo la misma proposición FORMAT, pero empezando un nuevo registro, es decir, otra tarjeta. Luego, los valores para c y D se leen de la tarjeta 2 en las columnas 1-8 y 11-25, respectivamente. (Véase también la sección 10.6.)

Similarmente, las proposiciones

```
      WRITE(6, 30) A, B, C, D
30    FORMAT(5X, F8.2, 2X, E14.7)
```

producirán dos (2) cadenas de salida:

$$_{bbbbb} + XXXX.XX_{bb} + 0.XXXXXXXE + XX$$
$$_{bbbbb} + XXXX.XX_{bb} + 0.XXXXXXXE + XX$$

Luego, A y B serán impresos en una línea de acuerdo a las especificaciones F8.2 y E14.7, respectivamente, con cuatro espacios de C y D se imprimirán sobre la siguiente línea en forma similar.

Nota: En el caso inverso, donde hay más especificaciones de campo que variables, la ejecución de una proposición READ/WRITE se completa una vez que todas las variables han sido definidas/impresas, y, por consiguiente, se ignoran las especificaciones extras. Por ejemplo, si la proposición READ anterior se cambia a

```
READ(5, 20) A, B, C, D, E
```

entonces se leen tres tarjetas y a la variable E se le asigna el número que está en las columnas 1-8 de la tercera tarjeta, usando la especificación F8.2.

Caso B

Tambien se pueden usar diagonales (/) para definir registros múltiples. Los diagonales se usan en las proposiciones FORMAT, para indicar el fin de los registros. Así,

```
      READ(5, 40) A, B, C, D
40    FORMAT(F10.2, F10.2/E15.7, E15.7)
```

define dos registros; esto es, leerá A y B utilizando F10.2 de la tarjeta 1 y C, D utilizando E15.7 de la tarjeta 2. En la misma forma,

```
      WRITE(6, 50) N, AVE
50    FORMAT('1','NUMERO DE ESTUDIANTES=',I3/1X,'PROMEDIO EXAMEN=',F7.2)
```

define dos registros, e imprimirá en el comienzo de una nueva página las dos líneas siguientes:

```
NUMERO DE ESTUDIANTES   XXX
PROMEDIO EXAMEN   XXX.XX
```

Hacemos énfasis en el hecho de que para cada nuevo registro de salida, debe proveerse el control de carro. Más exactamente, la especificación 1X que sigue a la diagonal en la proposición FORMAT, es la instrucción de control de carro para el nuevo registro.

Pueden aparecer también diagonales consecutivas. Sin embargo, en entrada las diagonales que aparecen al finalizar una proposición FORMAT dan un resultado ligeramente diferente del caso en que se pusiera la diagonal en medio de la proposición FORMAT. Específicamente:

1. *Salida.* Cuando aparecen n diagonales consecutivas, entonces se imprimen $n-1$ líneas en blanco. Por ejemplo,

```
      WRITE(6, 10) J, K                    WRITE(6, 20) J, K
10    FORMAT(1X, I10///1X, I10)     y      20 FORMAT(1X, I10///)
```

   darán como resultado dos líneas en blanco que aparecen entre J y K.

2. *Entrada.* Cuando aparecen en medio n diagonales consecutivas, entonces se saltan $n-1$ tarjetas; pero cuando aparecen al final n diagonales consecutivas, entonces se saltan n tarjetas. Por ejemplo,

```
      READ(6, 30) J, K
30    FORMAT(I10///I10)
```

   hará que se lea J en la primera tarjeta y K en la cuarta; es decir, se saltaron dos tarjetas. Sin embargo,

```
      READ(6, 40) J, K
40    FORMAT(I10///)
```

   hará que se lea J en la primera tarjeta y K en la quinta; esto es, se saltaron tres tarjetas.

(Ver también problemas 3.39 y 3.40.)

3.9 FACTOR DE REPETICION

Supongamos que se van a leer e imprimir 4 números A, B, C y D utilizando la misma especificación F10.2. La proposición FORMAT sería:

```
FORMAT(F10.2, F10.2, F10.2, F10.2)
```

Para simplificar, se puede utilizar un factor de repetición de 4:

```
FORMAT(4F10.2)
```

Esas dos proposiciones FORMAT tienen exactamente el mismo significado. Similarmente,

```
FORMAT(2I4, 3E15.7)
```

y

```
FORMAT(I4, I4, E15.7, E15.7, E15.7)
```

son equivalentes.

Se puede, también, repetir un grupo de especificaciones de campo. Por ejemplo,

```
FORMAT(5X, 2(I5, 3X, F8.2, 2X))
```

significa

```
FORMAT(5X, I5, 3X, F8.2, 2X, I5, 3X, F8.2, 2X)
```

y

```
FORMAT(3(3X, I4), 2(5X, E9.2))
```

es lo mismo que

```
FORMAT(3X, I4, 3X, I4, 3X, I4, 5X, E9.2, 5X, E9.2)
```

Estas abreviaciones permiten, a nuestras proposiciones FORMAT, ser más compactas y más fáciles de escribir.

3.10 UN PROGRAMA COMPLETO Y SENCILLO

Hemos cubierto suficiente material para escribir algunos programas completos y sencillos. Vamos a escribir uno aquí y a proponer algunos en la sección de problemas resueltos.

Supongamos que las longitudes de los lados de un triángulo T son a, b y c. El perímetro P de T es la suma de esas longitudes:

$$P = a + b + c$$

El área de T es dado por la fórmula:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

donde $s = P/2 = (a + b + c)/2$.

Supongamos que deseamos calcular el perímetro y el área de un triángulo cuyos lados tienen longitudes de 38.6, 42.4 y 56.1. Primero perforamos esos números en una tarjeta de datos utilizando, por ejemplo, campos de longitud 10, como en la figura 3-11.

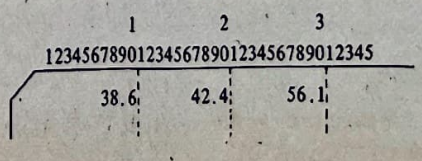


Fig. 3-1

El programa Fortran que ejecuta los anteriores cálculos es como sigue:

```
C
C       PROGRAMA QUE CALCULA EL PERIMETRO Y EL AREA DE UN TRIANGULO
C
        READ(5, 11) A, B, C
   11   FORMAT(3F10.1)
        P = A + B + C
        S = P/2.0
        AREA = SQRT(S*(S - A)*(S - B)*(S - C))
        WRITE(6, 12) A, B, C, P, AREA
   12   FORMAT(3(F8.1, 4X), F10.1, 4X, F12.2)
        STOP
        END
```

Explicación del programa y ejecución

Líneas 1-3. Estas líneas forman un comentario, puesto que se perforó C en la primera columna.

Líneas 4-5. Esas líneas le dicen al computador que lea los valores de A, B y C de la tarjeta de datos, utilizando campos de longitud 10. Luego para A, B y C, se asignan los siguientes valores:

$$A \leftarrow 38.6, \qquad B \leftarrow 42.4, \qquad C \leftarrow 56.1$$

Línea 6. El computador suma el valor de A, B y C, que resulta ser 137.1 y le asigna este valor a P:

$$P \leftarrow 137.1$$

Línea 7. El computador asigna P/2 a S:

$$S \leftarrow 68.55$$

Línea 8. El computador evalúa la expresión aritmética del lado derecho de la proposición, donde SQRT significa raíz cuadrada y asigna este valor, que es 817.56570 a AREA:

$$AREA \leftarrow 817.56570$$

Líneas 9-10. El computador imprime los valores de A, B, C, P y AREA, utilizando campos de $8 - 1 = 7$, 8, 8, 10 y 12, respectivamente, como en la figura 3-12. Obsérvese que estos campos están separados por blancos de longitud 4 (4X). Nótese, además, que P está redondeado a una cifra decimal (F10.1) y AREA está redondeada a dos cifras decimales (F12.2).

Línea 11. Esta línea le dice al computador que pare (STOP).

Línea 12. La proposición END, le dice al compilador que no hay más proposiciones en el programa, para traducir. No se requerirá tiempo de ejecución en el programa.

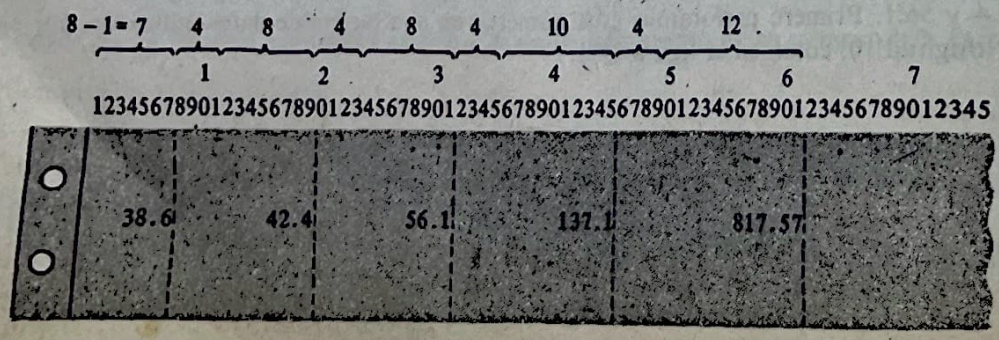


Fig. 3-12

Nota: en la ejecución del programa anterior, indicamos que el valor de A era 38.6. El número almacenado en A es:

0.38600000E+02

puesto que la representación interna de los valores reales está en forma exponencial, con aproximadamente 8-9 dígitos. (Ver apéndice A.)

Problemas resueltos

ENTRADA/SALIDA SIN FORMATO

3.1 Encontrar los errores, si los hay, en las siguientes proposiciones de E/S sin formato:

(*a*) READ A, B, X, Y
(*b*) PRINT NUMB, INT, RATE,
(*c*) READ, ID, WAGE, MAX, MIN
(*d*) PRINT, ROOT, COEF, RANGE,

(*a*) No hay coma después de READ.
(*b*) No hay coma después de PRINT, y sobra la coma después de RATE.
(*c*) Correcta.
(*d*) sobra la coma después de RANGE.

3.2 Supongamos que tres tarjetas de datos están perforadas como sigue:

Primera tarjeta: 22.2,3.33,444
Segunda tarjeta: 555,666,77.77
Tercera tarjeta: 8.888,9.99

Encontrar los valores asignados a las variables, si se ejecutan las siguientes proposiciones:

(*a*) READ A, B
READ, J, K, X, Y

(*b*) READ, A, B, J, K
READ, X, Y

(*a*) Cuando el computador ejecuta la primera proposición READ, asigna 22.2 a A y 3.33 a B. Esto completa la ejecución de la primera proposición READ; luego, se descarta la primera tarjeta aun cuando no se leyó el tercer valor. Cuando el computador ejecuta la segunda proposición READ, leerá la tarjeta que está encima del paquete de datos, es decir, la segunda tarjeta de datos. Luego, asigna 555 a J, 666 a K y 77.77 a X, entonces lee la siguiente tarjeta para asignar 8.888 a Y. Así, obtendremos:

A = 22.2, B = 3.33, J = 555, K = 666, X = 77.77, Y = 8.888

(*b*) La primera proposición READ nos conduce a:

A = 22.2, B = 3.33, J = 444, K = 555

La segunda proposición READ dará:

X = 8.888, Y = 9.99

Nótese que la segunda tarjeta de datos es descartada después de la ejecución de la primera proposición READ, por lo que es necesario usar la tercera tarjeta, para la ejecución de la segunda proposición READ.

3.3 Supongamos que se quieren asignar los valores 123, 456, 7.77, 8.88 y 9.99 a las variables L, M, X, Y y Z, respectivamente. Escriba una proposición READ sin formato y muestre cómo se deben perforar los datos en las tarjetas, si: (*a*) se usa solamente una tarjeta de datos, (*b*) se usan dos tarjetas de datos, la primera tarjeta con los valores enteros y la segunda con los valores reales.

En cada caso tendremos la proposición READ

```
READ, L, M, X, Y, Z
```

Los datos serán perforados como sigue:

(*a*) Primera tarjeta: 123, 456, 7.77, 8.88, 9.99
(*b*) Primera tarjeta: 123, 456
Segunda tarjeta: 7.77, 8.88, 9.99

3.4 Supongamos que se perfora una tarjeta de datos como sigue:

Tarjeta de datos: 123, 44.4

(*a*) Determinar la salida si se ejecuta el siguiente segmento de programa:

```
READ, J, X
K = J**2
Y = 3.0*X
PRINT, J, K, Y
```

(*b*) Explicar la diferencia que hay, si la única proposición PRINT se reemplaza por las dos proposiciones:

```
PRINT, J, K
PRINT, Y
```

(*a*) Ejecución del programa:

Línea 1. Se le asignan a J y a X los valores de la tarjeta de datos. Luego J = 123 y K = 44.4.

Línea 2. Se asigna a K el cuadrado de J; así K ← 15129.

Línea 3. Se asigna a Y tres veces el valor de X, así Y ← 133.2.

Línea 4. Se imprimen los valores de J, K y X, con el valor de Y, en forma exponencial normalizada. Luego, la salida aparecerá como sigue:

```
 123   15129   0.1332E 03
```

(*b*) Aquí los valores de J y K aparecen en una línea y el valor de Y en la siguiente línea, por lo que la salida apareceré como:

```
  123    15129
0.1332E 03
```

ENTRADA CON FORMATO

3.5 Localice los errores, si los hay, en cada proposición READ, así como su proposición FORMAT correspondiente.

(*a*)
```
     READ(5, 11) A, K, M, Z,
11   FORMAT(F8.0, I15, I10, I15)
```

(*b*)
```
     READ(5, 11) A, B, J, K, L
11   FORMAT(3F8.1, 2I8)
```

(*c*)
```
     READ(5, 11), A, B, C, D, J, K, L
11   FORMAT(4F15.2, 3I15)
```

(*a*) Primero, no debe haber coma después de Z. Segundo, la cuarta variable en la proposición READ es la variable real Z, pero su entrada correspondiente en la proposición FORMAT es la especificación de campo entero I15.

(*b*) Nótese primero que la proposición FORMAT es una abreviación de

```
11   FORMAT(F8.1, F8.1, F8.1, I8, I8)
```

En consecuencia, la variable entera J tiene una especificación de campo real F8.1.

(*c*) Primero, no debe haber coma antes de A. Segundo, la suma de las longitudes de campo es 105, lo que resulta muy grande para una tarjeta de datos que tiene solamente 80 columnas.

3.6 Supongamos que una tarjeta de datos está perforada como se muestra en la figura 3-13. Encuentre los valores de J, K y L que resultan de cada uno de los siguientes casos:

(*a*)
```
      READ(5, 11) J, K, L
11    FORMAT(I4, I3, I8)
```

(*b*)
```
      READ(5, 12) J, K, L
12    FORMAT(I3, 2X, I6, 3X, I3)
```

(*c*)
```
      READ(5, 13) J, K, L
13    FORMAT(2I6, I2)
```

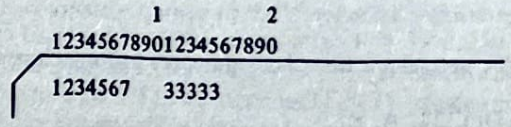


Fig. 3-13

Nótese que se leen solamente enteros para las variables, puesto que todas las especificaciones de campo son de la forma *Iw*, donde *w* es la longitud del campo.

(*a*) Las longitudes de campo son 4, 3 y 8, respectivamente. Entonces, a J se le asigna el entero perforado en las primeras 4 columnas (columnas 1-4), a K se le asigna el entero perforado en las tres columnas siguientes (columnas 5-7), y a L se le asigna el entero perforado en las 8 columnas siguientes (columnas 8-15). Luego J = 1234, K = 567, L = 33333.

(*b*) Las cinco especificaciones de campo instruyen al computador como sigue:

- I3 Asigna a J el entero perforado en las tres primeras columnas (columna 1-3), así que 123 se asigna a J.
- 2X Salta las 2 columnas siguientes (columna 4-5), aunque haya información perforada en esas columnas
- I6 Asigna a K el entero perforado en las seis columnas siguientes (columnas 6-11). Como los espacios en blanco se interpretan como ceros en los campos numéricos, se asigna 670003 a K.
- 3X Salta las tres columnas siguientes (columnas 12-14).
- I3 Asigna a L el entero perforado en las tres columnas siguientes (columnas 15-17), de modo que se asigna a L el valor de 300.

Luego, tendremos que J = 123, K = 670003, y L = 300.

(*c*) 2I6 repite dos veces a I6, es decir, las especificaciones de campo son I6, I6 e I2 con longitudes de campo de 6, 6 y 2, respectivamente. Esto resulta en J = 123456, K = 700033 y L = 33.

3.7 Supongamos que un programa tiene las siguientes proposiciones:

```
      READ(5, 21) A, B
21    FORMAT(F10.2, F12.3)
```

Encontrar los valores asignados a A y B si (*a*) se lee la tarjeta A de la figura 3-14, (*b*) se lee la tarjeta B de la figura 3-14.

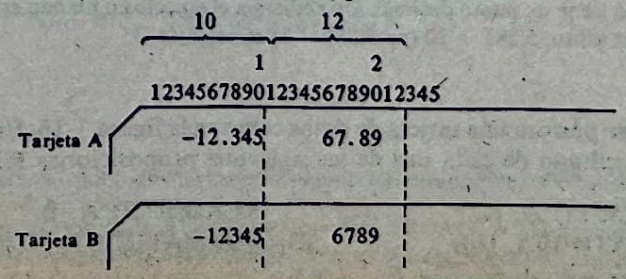


Fig. 3-14

Nótese primero que las longitudes de campo son 10 y 12 respectivamente; luego a A se le asigna el número real que se encuentre en las columnas 1-10 y a B se le asigna el número real que se encuentre en las columnas 11-22.

(*a*) Puesto que −12.345 está perforado en el primer campo (columna 1-10), este valor se asigna a A y como 67.89 está perforado en el segundo campo (columna 11-22), se asigna a B. Ambos números tienen puntos decimales en la tarjeta, luego no se tienen en cuenta el 2 de F10.2 y el 3 de F12.3

(*b*) Como no hay punto decimal perforado en el primer campo (columna 1-10), entonces, la especificación de campo F10.2 causará que se encaje un punto decimal después de dos lugares, desde el lado derecho del campo, es decir, entre las columnas 8 y 9. Luego, −123.45 se asigna a A. El número perforado en el segundo campo no tiene punto decimal, así que F12.3 causará que se encaje un punto decimal después de tres lugares a la derecha del campo, es decir, entre las columnas 19 y 20. Luego, 678.9 se asigna a B. Hacemos énfasis en el hecho de que la posición de un número en un campo puede hacerlo diferente si no está perforado con un punto decimal.

3.8 Supongamos que un programa tiene las siguientes proposiciones:

```
      READ(5, 22) A, B
22    FORMAT(E12.4, E15.6)
```

Encontrar los valores asignados a A y B, si la tarjeta de datos es (*a*) la tarjeta (1) de la figura 3-15, (*b*) la tarjeta (2) de la figura 3-15.

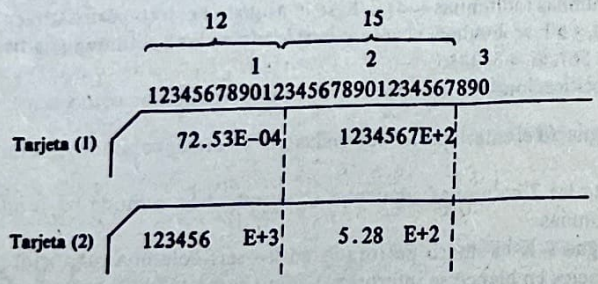


Fig. 3-15

Nótese que las longitudes de campo son 12 y 15, respectivamente. A es asignado al número real que se encuentra en las columnas 1-12 y B es asignado al número real que se encuentra en las 15 columnas siguientes, es decir, en las columnas 13-27.

(*a*) Nótese que 72.53E −04 está perforado en el primer campo (columna 1-12), luego es asignado a A. El 4, en la especificación de campo E12.4, no se tiene en cuenta, puesto que el número está perforado con un punto decimal. En el segundo campo, (columna 13-27), el número no está perforado con punto decimal. Entonces, el 6 de la especificación de campo E15.6, le dice al computador que coloque un punto decimal, seis (6) lugares (columnas) a la izquierda del carácter E, esto es, entre las columnas 8 y 9. Luego, 1.234567E + 2 se asigna a B.

(*b*) Nótese que no hay perforado un punto decimal en el primer campo; en consecuencia, el 4 de la especificación de campo E12.4 hará que se coloque un punto decimal entre la 4a. y 5a. columnas a la izquierda del carácter E, esto es, entre las columnas 5 y 6. Luego, se asigna 12345.6E + 3 a A. El número perforado en el segundo campo tiene un punto decimal. Sin embargo, el espacio en blanco en la columna 27 se interpreta como cero; por tanto, 5.28E + 20 (no 5.28E + 2) se asigna a B.

3.9 Supongamos que se perfora una tarjeta de datos como en la figura 3-16. Encontrar los valores de las variables como resultado de cada una de las siguientes proposiciones READ/FORMAT:

(*a*)
```
      READ(5, 21) A, K
21    FORMAT(F10.1, I10)
```

(*b*)
```
      READ(5, 22) A, B
22    FORMAT(F10.3, F10.6)
```

(*c*)
```
      READ(5, 23) A, B
23    FORMAT(E10.3, E10.4)
```

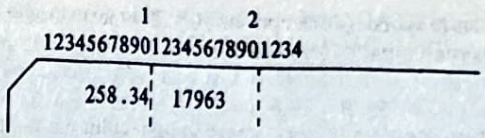


Fig. 3-16

(*a*) Las dos especificaciones de campo contienen longitudes de campo de 10 y 10, respectivamente, e instruyen al computador como sigue:

F10.1 Asigna a A el número real perforado en las 10 primeras columnas (columnas 1-10); luego, A = 258.34.(El 1 en F10.1 no se tiene en cuenta, puesto que el número tiene un punto decimal en el campo.)

I10 Asigna a K el entero perforado en las 10 columnas siguientes (columnas 11-20), donde los espacios en blanco se leen como ceros; luego K = 17963000.

(*b*) Como en la parte (*a*), obtenemos A = 258.34. La segunda especificación de campo le dice al computador que asigne a B el número real que está en las columnas 11-20. Puesto que no hay punto decimal dentro del campo, el 6 de F10.6 le dice al computador que coloque un punto decimal seis espacios a la izquierda de la última columna (columna 20). Entonces, obtenemos B = 17.963.

(*c*) Obsérvese primero, que no aparece el carácter E en la tarjeta de datos; de tal forma el campo *Ew.d* se trata lo mismo que la especificación de campo *Fw. d*. Los dos campos contienen longitudes de campo de 10 y 10, respectivamente, e instruyen al computador como sigue:

E10.3 Asigna a A el número real perforado en las primeras 10 columnas (columnas 1-10). Puesto que el campo tiene un punto del decimal, 258.34 es asignado a A.

E10.4 Asigna a B el número real perforado en las 10 columnas siguientes (columnas 11-20). Puesto que no hay punto decimal dentro del campo, se coloca un punto decimal cuatro espacios a la izquierda de la última columna. Esto da B = 1796.3

3.10 Supongamos que las primeras tres tarjetas, de un paquete de datos, están perforadas como en la figura 3-17. Encontrar los valores de las variables para cada uno de los siguientes casos:

(*a*)
```
      READ(5, 11) J, A
      READ(5, 11) K, B, L, M
11    FORMAT(I5, F5.2, I5)
```

(*b*)
```
      READ(5, 12) J, K, A, L, B, M
12    FORMAT(I5/I5, F5.2/I5, F5.2, I5)
```

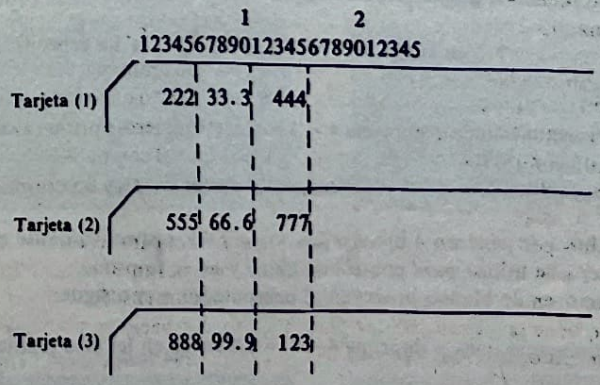


Fig. 3-17

Obsérvese primero, que todas las especificaciones de campo tienen longitudes de campo de 5.

(*a*) Recuérdese que la ejecución de cualquier proposición READ/WRITE no se completa hasta que todas las variables listadas tengan valores asignados/impresos. La primera proposición READ lee de la tarjeta 1, y luego las variables J y A se leen con especificaciones I5 y F5.2, respectivamente. Entonces, J = 222 y A = 33.3. La ejecución de esta proposición READ ha quedado completa; luego, la tarjeta 1 es descartada.

La segunda proposición READ también empieza leyendo de la tarjeta que está ahora encima, es decir, la tarjeta 2. Como hay más variables (4) que especificaciones de campo, se leen registros múltiples. En este caso, se leen dos tarjetas. Las variables K, B, L se leen de la tarjeta 2, utilizando I5, F5.2, respectivamente, lo que nos conduce a K = 555, B = 66.6, I = 777. La variable M se lee de la tarjeta 3, utilizando I5, de donde M = 888. Como todas las variables tienen ahora valores asignados, queda completa la ejecución de la segunda proposición READ, y la tercera tarjeta es descontada.

(*b*) Como hay dos diagonales, se leen al menos 3 registros (tarjetas). La variable J se lee de la tarjeta 1 con I5; entonces J = 222. Las variables K y A se leen de la tarjeta 2 con I5 y F5.2, respectivamente; entonces, K = 555 y A = 66.6. Finalmente, de la tarjeta 3 deducimos que L = 888, B = 99.9, M = 123.

SALIDA CON FORMATO

3.11 Localice los errores, si los hay, en cada par WRITE-FORMAT:

```
(a)      WRITE(6, 31), A, B, N
     31  FORMAT(F10.2, 3X, I8, 5X, I6)

(b)      WRITE(6, 32) J, Z, K,
     32  FORMAT(1X, I10, 4X, 2F10.3)
```

(Suponer que ninguna proposición de tipo ha cambiado los tipos de cualquiera de las variables.)

(*a*) Hay una coma antes de A. La variable real B tiene la especificación de campo entera I8.

(*b*) Hay una coma después de K; además, la variable entera K tiene la especificación de campo real F10.3.

3.12 Supongamos que K contiene 12345 y que ejecuta la proposición WRITE

```
WRITE(6, 41) K
```

Describa la salida, si la proposición FORMAT que la acompaña es

```
(a)  41  FORMAT(I10)          (c)  41  FORMAT(4X, I3)
(b)  41  FORMAT(4X, I8)       (d)  41  FORMAT(I5)
```

(*a*) La especificación de campo tiene una longitud de 10; por lo tanto, el valor de K será impreso justificado a la derecha, en las primeras 10 − 1 = 9 columnas de la página de impresión como en la figura 3-18(*a*). En otras palabras, el valor de K es generado justificado a la derecha en los primeros 10 caracteres de la cadena de salida, esto es, se genera ␣␣␣␣␣12345; pero el primer carácter en blanco se usa para control de carro y no se imprime.

(*b*) La salida aparecerá como en la figura 3-18(*b*), puesto que las dos especificaciones de campo instruyen al computador como sigue:

```
4X  Avance una línea e imprima 4 − 1 = 3 blancos en las primeras tres columnas de la página de
    impresión.
I8  Imprima el valor de K justificado a la derecha en las 8 columnas siguientes (columnas 4-1).
```

En otras palabras, se generan 4 blancos ␣␣␣␣ y ␣␣␣12345, respectivamente en la cadena de salida, pero el primer carácter ␣ se utiliza para control de carro y no se imprime.

(*c*) Las especificaciones de blanco instruyen al computador como sigue:

```
4X  Avance una línea e imprima 4 − 1 = 3 blancos en las tres primeras columnas de la página de
    impresión.
I3  Imprima el valor de K justificado a la derecha en las tres columnas siguientes (columnas 4-6).
```

Puesto que la longitud de campo para K es menor que el número de dígitos en K, se imprimirán tres asteriscos *** como en la figura 3-18(*c*) o se truncará el valor de K sobre la izquierda, de modo que se imprimiría solamente 345.

(*d*) La longitud de campo de I5 es igual al número de dígitos en K, luego se genera 12345 en la cadena de salida. Sin embargo, el primer carácter "1" se utiliza para control de carro y éste le dice a la impresora que avance al comienzo de la siguiente página antes de imprimir cuatro caracteres en las primeras 5 − 1 = 4 columnas de la página de impresión, como en la figura 3-18(*d*).

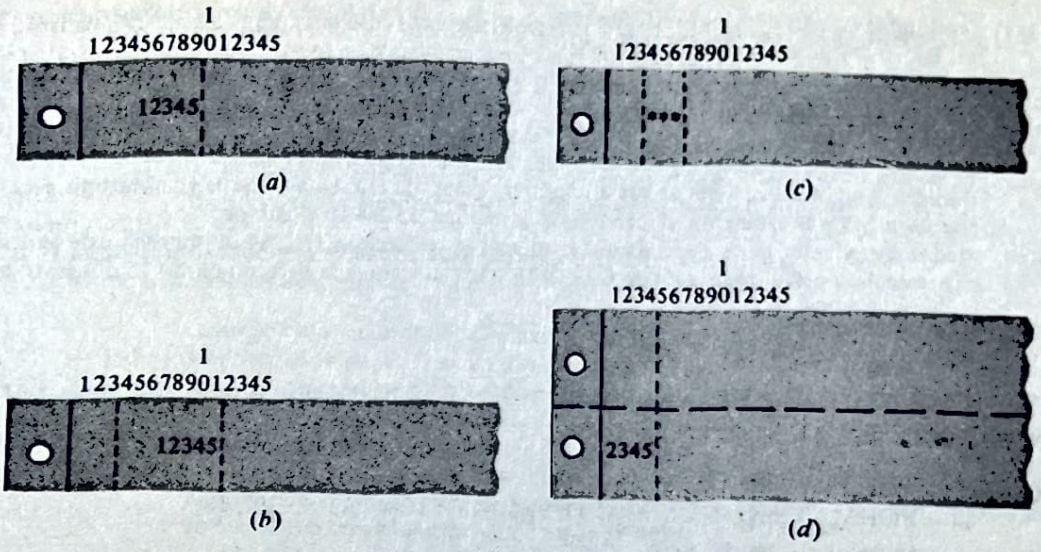


Fig. 3-18

3.13 Supongamos que A contiene el número 135.2837 y que se ejecuta la proposición WRITE

```
WRITE(6, 42) A
```

describa la salida, si la proposición FORMAT que la acompaña es:

(*a*) 42 FORMAT(6X, F10.1) (*c*) 42 FORMAT(6X, F6.3)
(*b*) 42 FORMAT(6X, F12.2) (*d*) 42 FORMAT(6X, E15.6)

En cada caso, la especificación de campo 6X hace que la impresora avance una línea e imprima blancos en las primeras $6 - 1 = 5$columnas (columnas 1-5) de la página de impresión. La salida aparece como en la figura 3-19, puesto que la segunda especificación de campo instruye al computador como sigue:

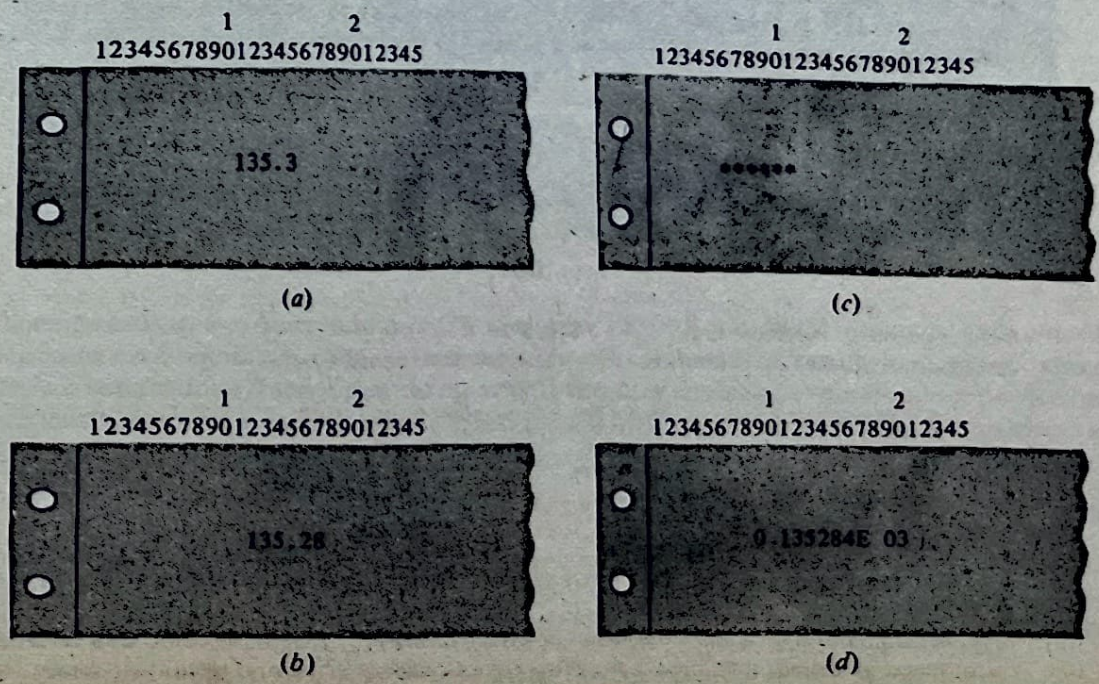


Fig. 3-19

(*a*) Redondee el valor de A a una cifra decimal, es decir, a 135.3 e imprima este número justificado a la derecha en las 10 columnas siguientes (columnas 6-15).

(*b*) Redondee el valor de A a dos cifras decimales, es decir, a 135.28, e imprima este número justificado a la derecha en las 12 columnas siguientes (columnas 6-17).

(*c*) Redondee el valor de A a tres cifras decimales, es decir, a 135.284, e imprima este número justificado a la derecha en las 6 columnas siguientes (columnas 6-11). Puesto que la longitud de campo 6 es menor que el número 7 de caracteres que se van a imprimir, se imprimirán seis asteriscos en el campo o el número se truncará sobre la izquierda, de modo que se imprime 35.284 en el campo.

(*d*) Redondee el valor de A a seis dígitos significativos, es decir, a 135.284, e imprima este valor en forma exponencial normalizada, esto es, 0.135284E 03, justificado a la derecha en las 15 columnas siguientes (columnas 6-20).

3.14 Supongamos que ID = 112233 y WAGE = 275.50. Determinar la salida para cada par WRITE-FORMAT:

(*a*)
```
      WRITE(6, 11) ID
11    FORMAT(6X, 'NUMERO DE IDENTIFICACION', 2X, I8)
```

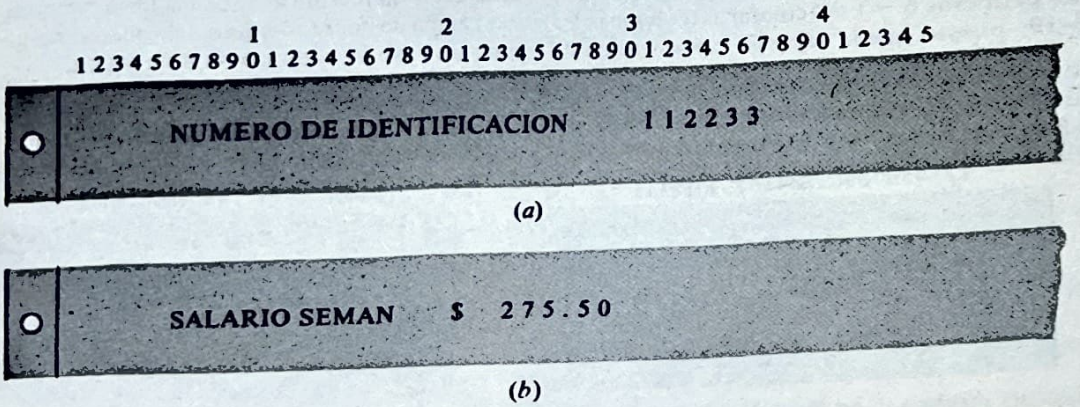
(*b*)
```
      WRITE(6, 12) WAGE
12    FORMAT(6X, 'SALARIO SEMAN $', F8.2)
```

(*a*) La impresora avanza una línea e imprime blancos en las primeras $6-1=5$ columnas (columnas 1-5) de la página de impresión. La entrada

'NUMERO IDENTIFICACION'

hace que se imprima el mensaje entre apóstrofes, en el campo siguiente. Puesto que el mensaje ocupa 21 espacios, se imprimirá en las 21 columnas siguientes (columnas 6-26). 2X hace que se impriman blancos en las columnas 27 y 28. I8 hace que el valor de ID se imprima justificado a la derecha, en las 8 columnas siguientes. Ver figura 3-20(*a*).

(*b*) La salida aparece como en la figura 3-20(*b*).



(*a*)

(*b*)

Fig. 3-20

3.15 Supongamos que ID = 25367, M = 138, A = 256.174 y B = −0.00285293. Encontrar la salida, si se ejecuta el par WRITE-FORMAT:

(*a*)
```
      WRITE(6, 11) ID, M, A, B
11    FORMAT(' ', I10, I6, F8.1, F10.3)
```

(*b*)
```
      WRITE(6, 12) ID, A, B
12    FORMAT('1', I10, 2E15.5)
```

(*a*) El primer símbolo en la cadena de salida es un blanco dado por ' '. Este actúa como una señal de control de carro; luego, se avanza una línea. Las cuatro especificaciones de campo restantes tienen longitudes de campo de 10, 6, 8 y 10 respectivamente, e instruyen al computador como sigue:

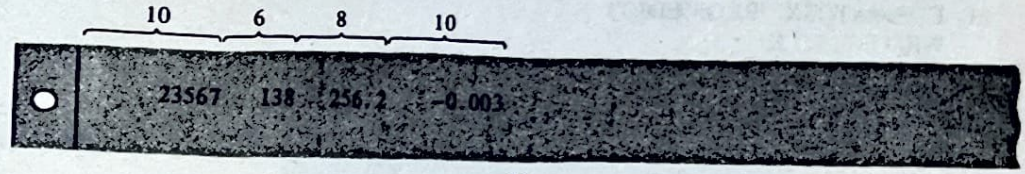
I10 Imprime el valor entero de ID justificado a la derecha, en las primeras 10 columnas de la página de impresión.
I6 Imprime el valor de M justificado a la derecha en las 6 columnas siguientes (columnas 11-16).
F8.1 Redondea el valor de A a una cifra decimal, esto es, 256.2, e imprime este número real justificado a la derecha en las 8 columnas siguientes (columnas 17-24).
F10.3 Redondea el valor de B a tres cifras decimales, esto es, −0.003, e imprime este número real justificado a la derecha, en las 10 columnas siguientes (columnas 25-34).

La salida aparece como en la figura 3-21(*a*).

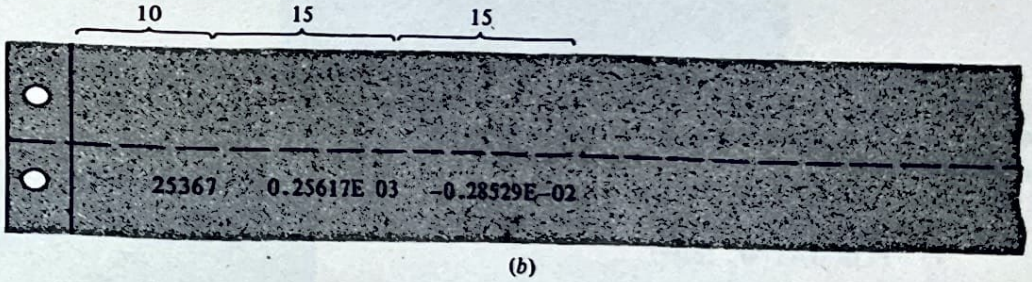
(*b*) Las especificaciones de campo instruyen al computador como sigue:

'1' Instrucción de control de carro para avanzar a una nueva página.
I10 Imprime el valor de ID en las diez primeras columnas de la página de impresión.
E15.5 Redondea el valor de A a cinco dígitos significativos, esto es, 256.17, e imprime este número en forma exponencial normalizada, esto es, 0.25617E 03, justificado a la derecha, en las 15 columnas siguientes (columnas 11-25).
E15.5 Redondea el valor de B a cinco dígitos significativos, esto es, −0.0028529, e imprime este número en forma exponencial normalizada, esto es, −0.28529E−02, justificado a la derecha, en las 15 columnas siguientes (columnas 36-40).

La salida aparece como en la figura 3-21(*b*).



(*a*)



(*b*)

Fig. 3-21

3.16 **Supongamos que se quieren imprimir los valores de J, A, K y B, de tal forma que cada entero quede colocado en 4 columnas, cada real esté colocado en 8 columnas con tres decimales y todos los valores queden separados al menos por tres espacios. Encontrar el rango de valores que se pueden imprimir para cada variable y escribir un par WRITE-FORMAT que (*a*) imprima los valores en una línea; (*b*) imprima los valores de J y A en una línea y los de K y B en la línea siguiente.**

Los valores de J y K deben estar entre −999 y 9999, y los valores de A y B entre −999.999 y 9999.999.

(*a*)
```
      WRITE(6, 11) J, A, K, B
11    FORMAT(1X, I4, 3X, F8.3, 3X, I4, 3X, F8.3)
```
ó
```
11    FORMAT(1X, 2(I4, 3X, F8.3, 3X))
```

(*b*)
```
      WRITE(6, 11) J, A, K, B
11    FORMAT(1X, I4, 3X, F8.3)
```

3.17 Supongamos que J1, J2 y J3 contienen tres notas de prueba y K contiene su promedio.

(*a*) Escribir un segmento de programa que imprima una nueva página lo siguiente, empezando en la columna 11:

```
NOTAS DE PRUEBA
    XXX
    XXX
    XXX
PROMEDIO
    XXX
```

donde los XXX denotan los valores de las notas de prueba y el promedio.

(*b*) Realizar lo anterior con una sola proposición WRITE.

(*a*)
```
      WRITE(6, 11)
11    FORMAT('1', 10X, 'NOTAS DE PRUEBA')
      WRITE(6, 12) J1, J2, J3
12    FORMAT(1X, I17)
      WRITE(6, 13)
13    FORMAT(13X, 'PROMEDIO')
      WRITE(6, 12) K
```

(*b*)
```
      WRITE(6, 21) J1, J2, J3, K
21    FORMAT('1', 10X, 'NOTAS DE PRUEBA'/1X, I17/1X, I17/1X, I17/13X,
                                                    'PROMEDIO'/1X, I17)
```

ó

```
21    FORMAT('1', 10X, 'NOTAS DE PRUEBA'/3(1X, I17/)13X, 'PROMEDIO'/1X, I17)
```

3.18 Supongamos que J1 = 111, J2 = 222, J3 = 333, J4 = 444 y J5 55. Determinar la salida, si se ejecuta el siguiente par de proposiciones:

```
      WRITE(6, 15) J1, J2, J3, J4, J5
15    FORMAT('0', I8/I9)
```

El computador avanza dos líneas, es decir, salta una línea ('0'), e imprime J1 en las primeras 8 columnas (I8). La diagonal / señala el fin del registro; de tal forma que J2 se imprime en las primeras 9 – 1 = 8 columnas (I9) de la siguiente línea, la proposición FORMAT se repite para J3 y J4 y se repite de nuevo para J5. El '0' hace que una línea se salte entre J2 y J3 y entre J4 y J5. La salida aparecerá como en la figura 3-22.

Fig. 3-22

PROGRAMAS SENCILLOS Y COMPLETOS

3.19 El área de superficie SUR y el volumen VOL de una caja con dimensiones *a, b, c* están dados por:

$$SUR = 2(ab + ac + bc) \quad \text{y} \quad VOL = abc$$

Supongamos que las dimensiones de una caja son 23.5, 41.3 y 16.2 y que están perforadas sobre una tarjeta de datos como en la figura 3-23(*a*). Encontrar la salida del siguiente programa:

```
C
C       AREA SUPERFICIAL Y VOLUMEN DE UNA CAJA
C
        READ(5, 61) A, B, C
   61   FORMAT(3F10.2)
        SUR = 2.0(A*B + A*C + B*C)
        VOL = A*B*C
        WRITE(6, 62) A, B, C, SUR, VOL
   62   FORMAT(6X, 'DIMENSIONES DE LA CAJA'/7X, 3(F5.1, 1X)//
     1          6X, 'AREA SUPERFICIAL =', F10.1//6X,'VOLUMEN=', F16.2)
        STOP
        END
```

```
         1         2         3
12345678901234567890123456789012345
        23.5      41.3      16.2
```

(*a*)

```
         1         2         3
12345678901234567890123456789012345
      DIMENSIONES DE LA CAJA
        23.5  41.3  16.2

      AREA SUPERFICIAL  4040.6

      VOLUMEN =        15722.91
```

(*b*)

Fig. 3-23

Ejecución del programa

El computador lee los valores de A, B y C de acuerdo a la proposición FORMAT numerada con 61, que es una abreviación de

```
   61   FORMAT(F10.2, F10.2, F10.2)
```

En cada caso, la longitud de campo es 10; de tal forma que a A, B y C, se les asignan los valores perforados en las columnas 1-10, columnas 11-20 y en las columnas 21-30, respectivamente. Luego, A = 23.5, B = 41.3, C = 16.2. (No se tiene en cuenta el 2 de F10.2, puesto que todos los números de la tarjeta de datos tienen punto decimal.)

El computador calcula 2.0*(A*B + A*C + B*C), que resulta 4040.62 y asigna este número a SUR, luego calcula A*B*C que resulta 15722.910 y asigna este número a VOL:

SUR ← 4040.62, VOL ← 15722.910

El par WRITE-FORMAT instruye al computador para que imprima la salida como en la figura 3-23(*b*). Obsérvese que una diagonal / hace que la impresora avance a la siguiente línea, pero que dos diagonales // hacen que la impresora avance dos líneas y, entonces, salte una línea. También, obsérvese que SUR está redondeado a una cifra decimal (F10.1) y que VOL está redondeado a dos cifras decimales (F16.2).

3.20 Suponiendo que $D = a_1b_2 - a_2b_1 \neq 0$, la solución de las ecuaciones lineales

$$a_1x + b_1y = c_1$$
$$a_2x + b_2y = c_2$$

está dada por

$$x = \frac{b_2c_1 - b_1c_2}{a_1b_2 - a_2b_1} \qquad y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1}$$

Escriba un programa FORTRAN que lea de una tarjeta de datos los valores de A1, B1, C1 y de una segunda tarjeta los valores de A2, B2 y C2; calcule el par de soluciones X, Y, y luego imprima la salida de tal forma que aparezca como sigue:

```
COEFICIENTES DE ECUACIONES LINEALES
    PRIMERO   *****    *****    *****
    SEGUNDO   *****    *****    *****
SOLUCION DE LAS ECUACIONES LINEALES
    X = ********
    Y = ********
```

```
C
C       SOLUCION DE ECUACIONES LINEALES
C
        READ(5, 11) A1, B1, C1, A2, B2, C2
   11   FORMAT(3F10.1)
        D = A1*B2 - A2*B1
        X = (B2*C1 - B1*C2)/D
        Y = (A1*C2 - A2*C1)/D
        WRITE(6, 12) A1, B1, C1, A2, B2, C2, X, Y
   12   FORMAT(11X, 'COEFICIENTES DE ECUACIONES LINEALES'//15X, 'PRIMERO',
     1          3X, 3F8.1/15X, 'SEGUNDO' 2X, 3F8.1//
     1          11X, SOLUCION DE LAS ECUACIONES LINEALES'//15X 'X =',
     1          2X, F12.2/15X, 'Y =', 2X, F12.2)
        STOP
        END
```

Problemas suplementarios

ENTRADA/SALIDA SIN FORMATO

3.21 Localizar errores, si los hay, en cada proposición de entrada diagonal salida sin formato:

(*a*) READ, FIRST, LAST, NEXT,
(*b*) PRINT ID, WAGE, RATE,
(*c*) READ, INT, LOT, AREA
(*d*) PRINT, A, B, C, D,

3.22 Supongamos que las primeras cuatro tarjetas de un paquete de datos se perforan como sigue:

Primera tarjeta: 11, 22, 3.3, 4.4
Segunda tarjeta: 5.5, 6.6, 77
Tercera tarjeta: 88, 99, 2.34
Cuarta tarjeta: 5.67, 8.90, 123, 456

Encontrar los valores asignados a las variables, si se ejectuan las siguientes proposiciones:

(*a*)
```
READ, J, K, A, B, C
READ, L, M, X, Y
```

(*b*)
```
READ, J, K, A
READ, B, C, L, M
READ, X, Y
```

(*c*)
```
READ, J, K, A
READ, B, C, L
READ, M
READ, X, Y
```

3.23 Supongamos que los valores 1.1, 2.2, 33, 44 y 55 han sido asignados a las variables A, B, J, K y L respectivamente, utilizando la proposición de lectura

```
READ, A, B, J, K, L
```

Demostrar cómo se perforarán los datos sobre la tarjeta, si

(*a*) Solamente se utiliza una tarjeta de datos.
(*b*) Se utilizan dos tarjetas de datos, la primera con los valores reales y la segunda con los valores enteros.

3.24 Supongamos que dos tarjetas han sido perforadas como sigue:

Primera tarjeta: 22.2, 15
Segunda tarjeta: 25, 35

Encontrar la salida, si se ejecuta el siguiente segmento de programa:

(*a*)
```
READ, A, J, K
B = A*6.0
PRINT, A, B, J, K
```

(*b*)
```
READ, A
READ, J, K
B = A*6.0
PRINT, A, B, J
PRINT, K
```

(*c*)
```
READ, A, J, K
B = A*6.0
PRINT, A
PRINT, B, J, K
```

(*d*)
```
READ, A
READ, J, K
B = A*6.0
PRINT, A, J
PRINT, B
PRINT, K
```

ENTRADA CON FORMATO

3.25 Localice los errores, si los hay, en cada par READ-FORMAT:

(*a*)
```
     READ(5, 10), X, Y, L, M,
10   FORMAT(F8.1, F9.2, 2I12)
```

(*b*)
```
     READ(5, 20) A, L, B, M
20   FORMAT(F10.3, I8)
```

(*c*)
```
     READ(5, 30) S, T, U, J, K
30   FORMAT(2F10.3, 3I8)
```

(*d*)
```
     READ(5, 40) A, J, B, K, C, L
40   FORMAT(F8.2, I6, F8.3)
```

3.26 Supongamos que una tarjeta de datos se perfora como se muestra en la figura 3-24 y supongamos, también, que se ejecuta la siguiente proposición de lectura:

```
READ(5, 50) J, K, L, M
```

Encuentre los valores asignados a J, K, L, M si la proposición FORMA que acompaña a la lectura es:

(*a*) `50 FORMAT(I2, I3, I8, I3)`
(*b*) `50 FORMAT(4I4)`
(*c*) `50 FORMAT(2I3, 5X, 2I3)`
(*d*) `50 FORMAT(3I3, 2I4)`

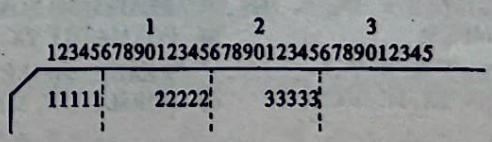


Fig. 3-24

3.27 Supongamos que se ejecutan las siguientes proposiciones:

```
     READ(5, 10) A, B, C
10   FORMAT(F10.2, F10.3, F10.2)
```

Encontrar los valores de A, B, C, si la tarjeta de datos es como la que se muestra en la figura 3-25.

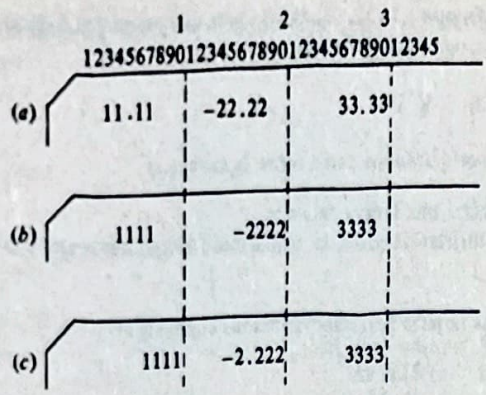


Fig. 3-25

3.28 Supongamos que se ejecutan las siguientes proposiciones:

```
      READ(5, 20) A, B, C
10    FORMAT(E10.2, E10.3, E10.2)
```

Encontrar los valores de A, B, C, si la tarjeta de datos es como la que se muestra en la figura 3-26.

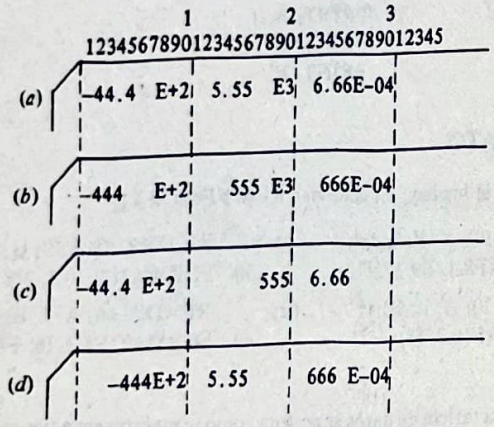


Fig. 3-26

3.29 Supongamos que una tarjeta de datos se perfora como en la figura 3-27. Encontrar los valores asignados a las variables, si se ejecuta el siguiente par de proposiciones:

(*a*)
```
      READ(5, 10) I, J, A, B
10    FORMAT(2I5, 2F5.1)
```

(*b*)
```
      READ(5, 20) A, B, C, D
20    FORMAT(4F5.3)
```

(*c*)
```
      READ(5, 30) J, K, X, Y, Z
30    FORMAT(I4, 2X, I4, 3F4.2)
```

(*d*)
```
      READ(5, 40) R, S, T
40    FORMAT(3F4.1)
```

(*e*)
```
      READ(5, 50) J, A, B, L, M
50    FORMAT(I3, 2X, 2F4.1, I3, I4)
```

(*f*)
```
      READ(5, 60) I, J, K, X, Y, Z
60    FORMAT(2X, 3I1, 3X, F4.2, 2F3.1)
```

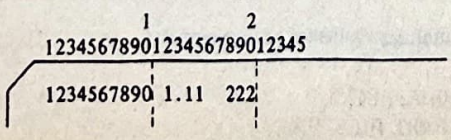


Fig. 3-27

3.30 Supongamos que las primeras tres tarjetas de un paquete de datos se perforan como en la figura 3-28. Encontrar los valores asignados a las variables, si se ejecutan las siguientes proposiciones:

(*a*)
```
      READ(5, 10) J, X
      READ(5, 10) K, Y, L, M
10    FORMAT(I5, F5.2, I5)
```

(*b*)
```
      READ(5, 20) J, A, B, K
      READ(5, 20) L, C
20    FORMAT(I5, 2F5.2)
```

(*c*)
```
      READ(5, 30) J, A, B
      READ(5, 30) K, X, Y, L, M
30    FORMAT(I5, 2F5.2, I5)
```

(*d*)
```
      READ(5, 40) J, X
      READ(5, 40) K, Y, Z
40    FORMAT(I5, 2F5.2)
```

(*e*)
```
      READ(5, 50) J, X, Y, Z, K, L, M
50    FORMAT(I5, 2F5.2/F5.2/I5, 5X, 2I4)
```

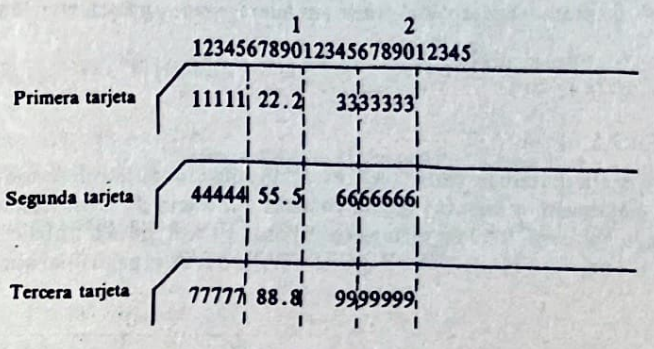


Fig. 3-28

SALIDA CON FORMATO

3.31 Localice errores, si los hay, en cada par de proposiciones WRITE-FORMAT:

(*a*)
```
      WRITE(6, 10) X, Y, M, N,
10    FORMAT(1X, 3F8.1, 2I9)
```

(*b*)
```
      WRITE(6, 20) A, L, B, M
20    FORMAT(1X, F9.2, 2(I8, E7.3))
```

(*c*)
```
      WRITE(6, 30), S, T, U, J, K
30    FORMAT(2E12.3, 2(F8.1, 3X, I7))
```

(*d*)
```
      WRITE(6, 40) A, J, B, C, K
40    FORMAT(F7.1, I8, E6.2)
```

3.32 Supongamos que J, K, L contienen los valores 1111, 2222, 3333, respectivamente, y que se ejecuta la proposición

```
WRITE(6, 50) J, K, L
```

Describa la salida, si la proposición FORMAT que la acompaña es la siguiente:

(*a*) `50 FORMAT(3I10)`
(*b*) `50 FORMAT(1X, I10/2I10)`
(*c*) `50 FORMAT(1X, 'LOS NUMEROS SON', 3I8)`
(*d*) `50 FORMAT(1X, I10)`
(*e*) `50 FORMAT(1X, 2I10)`
(*f*) `50 FORMAT(I4, 7X, I3, 7X, I4)`

3.33 Supongamos que A, B, C contienen 111.222, 444.666, 777.888, respectivamente, y que se ejecuta la proposición:

```
WRITE(6, 60) A, B, C
```

Describa la salida, si la proposición FORMAT que la acompaña es la siguiente:

(*a*) `60 FORMAT(1X, 3F10.2)`
(*b*) `60 FORMAT(1X, F10.1, 2X, F5.3, 2X, F10.1)`
(*c*) `60 FORMAT('1', E15.5, E15.2, E15.4)`
(*d*) `60 FORMAT(1X, 2E12.4)`
(*e*) `60 FORMAT(1X, F10.2///1X, E10.3///1X, F10.1)`
(*f*) `60 FORMAT(F7.3, 2X, E15.7/F15.1)`

3.34 Supongamos que se ejecuta la proposición

WRITE(6, 70) J, K, X, Y

Supongamos, también, que cada entero está localizado en cinco columnas y cada real en siete columnas con dos cifras decimales y que todos los valores están separados, por lo menos, por tres espacios. Encontrar el rango de valores que se puede imprimir para cada variable y escribir la proposición FORMAT que acompañaría a la proposición de impresión que:

(*a*) Imprima los valores sobre una línea.
(*b*) Imprima los valores sobre la primera línea de una nueva página.
(*c*) Imprima los valores sobre la quinta línea de una nueva página.
(*d*) Imprima los enteros sobre una línea y los reales en la siguiente línea.
(*e*) Imprima cada número sobre una línea diferente.
(*f*) Imprima los enteros en la tercera línea de una nueva página y los reales en la sexta línea de la misma página.

PROGRAMAS

3.35 Supongamos que el impuesto federal, FTAX, es el 15% y que el impuesto del estado, STAX, es del 2.5% del pago neto WAGE. Supongamos que una tarjeta contiene el número de identificación del empleado, ID, (nueve dígitos), su pago por hora, RATE y el número de horas, HOUR, que ha trabajado durante la semana. Escriba un programa que imprima ID, HOUR, WAGE, FTAX, STAX el pago final que recibiría el empleado, TPAY.

3.36 Supongamos que x, y, z son las siguientes funciones de t:

$$x = t^3 - 8t + 4, \qquad y = \operatorname{sen} t + \cos 2t, \qquad z = e^{3t+5}$$

Escribir un programa que lea t, x, y, z.

3.37 Supongamos que un automóvil parte del reposo y tiene una aceleración constante a por t segundos. La velocidad final v y la distancia d recorrida por el automóvil, son dadas por las fórmulas

$$d = \tfrac{1}{2}at^2 \qquad \text{y} \qquad v = at$$

Escribir un programa que lea a y t e imprima t, d y v.

3.38 Escribir un programa que lea la altura h y las longitudes de las dos bases b_1 y b_2 de un trapecio T e imprima h, b_1, b_2 y el área de T.

PROBLEMAS MISCELANEOS

3.39 Supongamos que se ejecuta la siguiente proposición de lectura:

READ(5, 10) A, B

Encontrar el número de tarjetas que se saltan antes de leer B, si la proposición FORMAT es:

(*a*) 10 FORMAT(F10.2////)
(*b*) 10 FORMAT(F10.2////F10.2)

[Generalmente, se saltan n tarjetas, si aparecen n diagonales al final de una proposición FORMAT, como en (*a*), pero solamente se saltan $n-1$ tarjetas, si las n diagonales aparecen en medio de la proposición FORMAT, como en (*b*).]

3.40 Supongamos que se ejecuta la siguiente proposición de lectura:

```
WRITE(5, 20) A, B
```

Encontrar el número de líneas en blanco, antes de que se imprima B, si la proposición FORMAT es:

(*a*) 20 FORMAT(1X, F10.2/////)
(*b*) 20 FORMAT(1X, F10.2/////1X, F10.2)

(Generalmente, se dejan $n-1$ líneas en blanco, cuando aparecen n diagonales al final o en medio de una proposición FORMAT. Comparar con el problema 3.39.)

Respuestas a problemas suplementarios seleccionados

3.21
(*a*) Lleva una coma después de NEXT.
(*b*) Debe tener una coma después de PRINT, pero no después de RATE.
(*c*) No hay errores.
(*d*) Lleva una coma después de D.

3.22
(*a*) 11, 22, 3.3, 4.4, 5.5, 88, 99, 2.34, 5.67
(*b*) 11, 22, 3.3, 5.5, 6.6, 77, 88, 5.67, 8.90
(*c*) 11, 22, 3.3, 5.5, 6.6, 77, 88, 5.67, 8.90

3.23
(*a*) Tarjeta de datos:
(*b*) Primera tarjeta:
Segunda tarjeta:

3.24
(*a*) 22.2, 133.2, 15, 25 en una línea.
(*b*) 22.2, 133.2, 25 en una línea y 35 en la siguiente línea.
(*c*) 22.2, en una línea y 133.2, 15, 25 en la línea siguiente.
(*d*) 22.2 y 25 en una línea, 133.2 en la línea siguiente y luego 35 en la siguiente.

3.25
(*a*) No lleva la coma antes de X ni después de M.
(*b*) No hay error.
(*c*) La variable real U corresponde al código de formato entero I8.
(*d*) La proposición FORMAT se repite, para K, C, L, pero los códigos de formato que aparecen no tienen el tipo correcto.

3.26
(*a*) 11, 111, 222, 220 (*c*) 111, 110, 222, 200
(*b*) 1111, 1000, 22, 2220 (*d*) 111, 110, 0, 222

3.27 (*a*) 11.11, −22.22, 33.33 (*b*) 11110.00, −2.222, 333.30 (*c*) 11.11, −2.222, 33.33

3.28
(*a*) −44.4E+2, 5.55E+3, 6.66E−4 (*c*) −44.4E20, 0.555, 6.66
(*b*) −4440.00E2, 55.500E3, 6.66E−4 (*d*) −4.44E2, 5.55, 66.60E−4

3.29
(*a*) 12345, 67890, 1.11, 22.2 (*c*) 1234, 7890, 1.1, 10.02, 22.00 (*e*) 123, 678.9, 1.0, 110, 222
(*b*) 12.345, 67.890, 1.11, 0.222 (*d*) 123.4, 567.8, 900.1 (*f*) 3, 4, 5, 90.01, 0.11, 0.2

3.30 (*a*) 11111, 22.2, 44444, 55.5, 66, 77777
(*b*) 11111, 22.2, 0.33, 44444, 77777, 88.8
(*c*) 11111, 22.2, 0.33, 44444, 55.5, 0.66, 66666, 77777
(*d*) 11111, 22.2, 44444, 55.5, 0.00
(*e*) 11111, 22.2, 0.33, 444.44, 77777, 9, 9999

3.31 (*a*) Hay una coma después de N. A la variable entera M le corresponde un código de formato real F8.1.
(*b*) No hay error.
(*c*) Hay una coma antes de S. A la variable entera K le corresponde el código del formato real F8.1
(*d*) No hay error.

3.32 (*a*) ␣␣␣␣1111␣␣␣␣␣␣2222␣␣␣␣␣␣3333
(*b*) ␣␣␣␣␣␣1111
␣␣␣␣␣␣2222␣␣␣␣␣␣3333
(*c*) LOS NUMEROS SON ␣␣␣␣1111␣␣␣␣2222␣␣␣␣3333.
(*d*) Los números aparecen en tres líneas diferentes.
(*e*) 1111 y 2222 aparecen en una línea y 3333 en la línea siguiente.
(*f*) No se da una señal explícita de control de carro. Así, el primer carácter de "1111" se usa para control de carro y solamente se imprimirán los tres restantes. Además, la longitud de campo para K es insuficiente; de tal forma, se imprimirá 222 ó ***. Entonces, en la primera línea de una nueva página aparecerá una de las dos siguientes:

111␣␣␣␣␣␣222␣␣␣␣␣␣3333 ó 111␣␣␣␣␣␣***␣␣␣␣␣␣3333

3.33 (*a*) ␣␣␣␣111.22␣␣␣␣444.67␣␣␣␣777.89
(*b*) ␣␣␣␣␣111.2␣␣*****␣␣␣␣␣␣777.9
(*c*) ␣␣␣␣0.11122E 03 ␣␣␣␣␣␣␣0.44E 03 ␣␣␣␣␣0.7779E 03 en la primera línea de una nueva página.
(*d*) ␣␣0.1112E 03 ␣␣0.4447E 03, y entonces ␣␣0.7779E 03 en la línea siguiente.
(*e*) ␣␣␣␣111.22 y entonces ␣0.445E 03 en la cuarta línea, y luego ␣␣␣␣␣777.9 en la séptima línea.
(*f*) 11.222 ␣␣␣␣0.4446660E 03 en la primera línea de una nueva página, y luego ␣␣␣␣␣␣␣␣777.9 en la línea siguiente.

3.34 J y K pueden imprimirse para valores que estén entre 99999 y −9999; y X y Y pueden imprimirse para valores entre 9999.99 y −999.99.

(*a*) FORMAT(1X, 2(I5, 3X), 2(F7.2, 3X))
(*b*) FORMAT('1', 2(I5, 3X), 2(F7.2, 3X))
(*c*) FORMAT('1'////1X, 2(I5, 3X), 2(F7.2, 3X))
(*d*) FORMAT(1X, I5, 3X, I5/1X, F7.2, 3X, F7.2)
(*e*) FORMAT(1X, I5/1X, I5/1X, F7.2/1X, F7.2)
(*f*) FORMAT('1'//1X, I5, 3X, I5///1X, F7.2, 3X, F7.2)

3.39 (*a*) 5 (*b*) 4

3.40 (*a*) 4 (*b*) 4

Capítulo 4

Transferencia de control. Diagramas de flujo

4.1 INTRODUCCION

Normalmente, el computador ejecuta las instrucciones de un programa Fortran una después de otra, a menos que se le instruya en otra forma. El orden de la ejecución puede controlarse por medio de varias instrucciones. Básicamente, hay dos tipos de transferencia de control: incondicional y condicional. Estas proposiciones de control se estudiarán principalmente en este capítulo, en el 5 y en el 12.

Cuando los programas se hacen más complejos, un diagrama de flujo es muy útil en la planeación, diseño y estructuración de un programa. Un *diagrama de flujo* es una representación gráfica de un algoritmo, es decir, es un esquema visual que nos da los pasos de un algoritmo y también el flujo de control entre los diferentes pasos. (Un *algoritmo* es un método para resolver un problema, paso por paso.) En particular, en un diagrama de flujo incluimos cada operación, instrucción, o serie de instrucciones en una caja; el flujo de control se indica por líneas dirigidas entre las cajas. Además, los diferentes tipos de operaciones se indican por diferentes formas de cajas, como se ilustra en la figura 4-1. Si un diagrama de flujo continúa en otra página, o si se dificulta la conexión de dos cajas, utilizamos un pequeño círculo rotulado para representar dicha conexión.

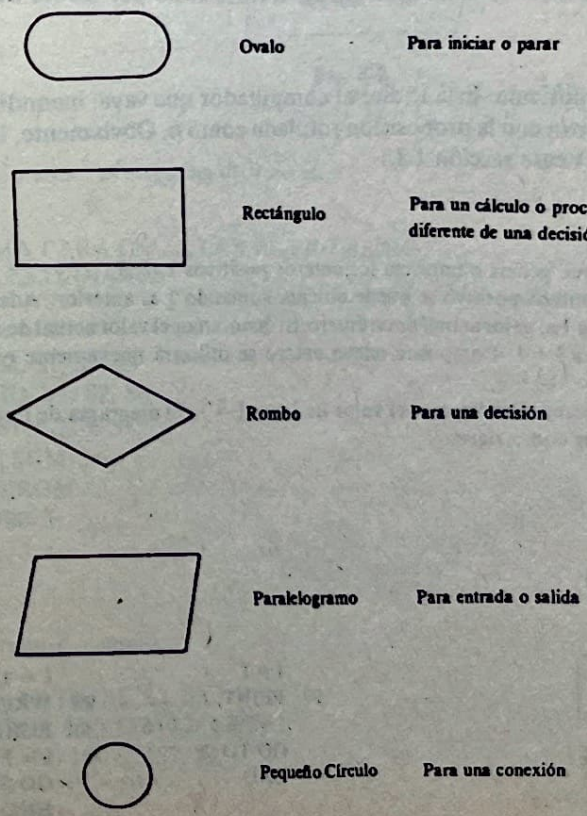
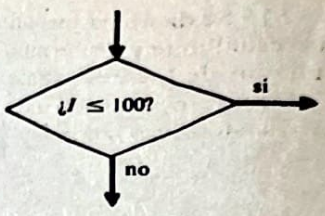
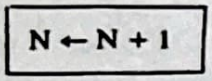


Fig. 4-1

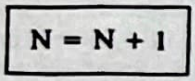
El rombo siempre denota una decisión y puede tener dos o más líneas que salen de él. Estas líneas se rotulan con los diferentes resultados de la decisión, es decir, con "sí" o "no", con "verdadero" o "falso", con "positivo", "negativo" o "cero". Por ejemplo, la pregunta "¿Es I ≤ 100?" puede aparecer en un diagrama de flujo como se muestra a la derecha. (Por conveniencia en la notación, usualmente omitiremos el símbolo de la pregunta.)



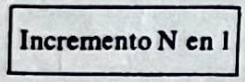
En este punto, mencionaremos una vez más que el símbolo aceptado universalmente para asignación es la flecha ←, pero ésta en Fortran se traduce por el signo =. Puesto que los diagramas de flujo son lenguajes independientes, muchos textos de Fortran prefieren utilizar la flecha en sus diagramas. En general, adoptaremos esta modalidad. Por ejemplo, para indicar que N se incrementa en 1, esto es, que N + 1 se asigna a N, usualmente utilizaremos



en lugar de



o de



Hacemos hincapié en que todo lo mencionado con anterioridad, se utiliza ampliamente en los textos de Fortran.

4.2 TRANSFERENCIA INCONDICIONAL

La transferencia incondicional de control se puede llevar a cabo por medio de la proposición

GO TO *n*

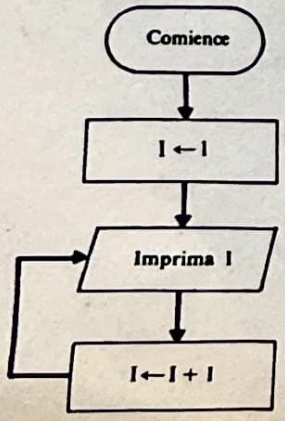
donde *n* es un número de proposición. Esta le dice al computador que vaya, incondicionalmente, a aquella parte del programa que comienza con la proposición rotulada como *n*. Obviamente, la proposición rotulada como *n* debe ser ejecutable. (Véase sección 1.8.)

EJEMPLO 4.1

(*a*) Escriba un programa que genere e imprima los enteros positivos 1, 2, 3,... .

Observemos que cualquier entero positivo se puede obtener sumando 1 al anterior. Además, éste es un proceso repetitivo, porque siempre se suma 1 al valor actual de un entero. Si denotamos el valor actual de un entero por I, entonces el próximo entero (su sucesor) es I + 1. Como este nuevo entero se utilizará nuevamente para calcular su sucesor, asignamos I + 1 a I.

A fin de comenzar este proceso, inicializamos el valor de I con I = 1. El diagrama de flujo de este programa y la traducción Fortran del proceso es como sigue:




```
      I = 1
99    PRINT, I
      I = I + 1
      GO TO 99
      END
```

ó

```
      I = 1
99    WRITE(6, 20) I
20    FORMAT(1X, I4)
      I = I + 1
      GO TO 99
      END
```

(*b*) Se da un paquete de tarjetas, cada una de las cuales contiene cuatro números que representan la identificación de un estudiante y los resultados de tres exámenes. Escriba un programa que calcule, para cada tarjeta, el promedio de los resultados de los tres exámenes.

De nuevo, el proceso de cálculo de los promedios de los tres números es un proceso repetitivo. Por esto se requiere una transferencia incondicional. El diagrama de flujo para este proceso se muestra en la figura 4-2.

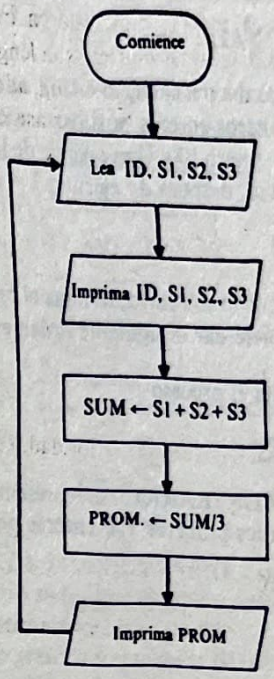


Fig. 4-2

El programa Fortran que traduce el diagrama de flujo es:

```
C
C       PROGRAMA PARA CALCULAR EL PROMEDIO
C       DE CADA ESTUDIANTE
C

  100   READ, ID, S1, S2, S3
        PRINT, ID, S1, S2, S3
        SUM = S1 + S2 + S3
        PROM   SUM/3.0
        PRINT, PROM
        GO TO 100
        END
```

Si se utiliza E/S con formato, tenemos:

```
  100   READ(5, 10) ID, S1, S2, S3
   10   FORMAT(I5, F10.2, F10.2, F10.2)
        WRITE(6, 10) ID, S1, S2, S3
        SUM = S1 + S2 + S3
        PROM = SUM/3.0
        WRITE(6, 20) PROM
   20   FORMAT(1X, 'EL PROMEDIO ES', F8.2)
        GO TO 100
        END
```

Obsérvese que en cada uno de los ejemplos anteriores, el proceso se repite sin parada varias veces; decimos, entonces, que cada programa tiene un *ciclo infinito*. Obviamente, para terminar el proceso apropiadamente, necesitamos alguna instrucción que permita ejecutar decisiones. Esto nos conduce a la sección siguiente. Recordemos que son precisamente estas capacidades de ejecutar decisiones, las que hacen poderosos a los computadores.

4.3 TRANSFERENCIA CONDICIONAL

Consideremos nuevamente el problema del ejemplo 4.1(*a*), en el cual se generan enteros. Supongamos que deseamos generar sólo los 100 primeros enteros positivos, es decir, supongamos que queremos que el programa termine después de generar el entero 100. Una manera de lograr esto es hacer la siguiente pregunta en el punto A de la figura 4-3(*a*). Esto es, después de ejecutar I = I + 1, preguntar:

¿Es el valor actual de I ≤ 100?

Si la respuesta es sí, se repite el proceso; de lo contrario, termina el proceso. En otras palabras, si se utilizan conexiones ordinarias en español, se quiere dar la siguiente orden en el punto A:

Si I ≤ 100 ENTONCES repita el proceso
DE LO CONTRARIO pare.

Esta orden SI...ENTONCES...DE LO CONTRARIO... se representa por el rombo de decisión de la figura 4-3(*b*). El diagrama de flujo que genera los primeros 100 enteros positivos se da en la figura 4-4.

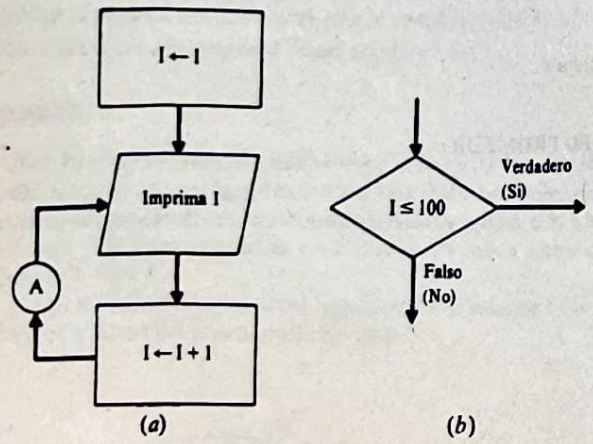


Fig. 4-3

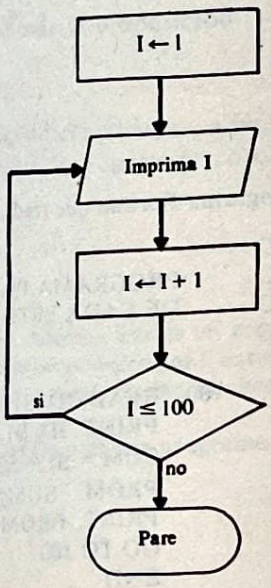


Fig. 4-4

Consideremos otro ejemplo. Supongamos que el monto, AMT, de un préstamo se perfora en una tarjeta y que la TASA de interés es 7% si AMT ≤ $10 000 y 6% si AMT es mayor que $10 000. Queremos escribir un segmento de programa que calcule el interés, INT. Después de leer AMT, deseamos dar la orden:

SI AMT ≤ 10 000 ENTONCES TASA = 0.07
DE LO CONTRARIO TASA = 0.06

De nuevo, tenemos la conexión SI...ENTONCES...DE LO CONTRARIO... El diagrama de flujo, para tal segmento de programa, aparece en la figura 4-5.

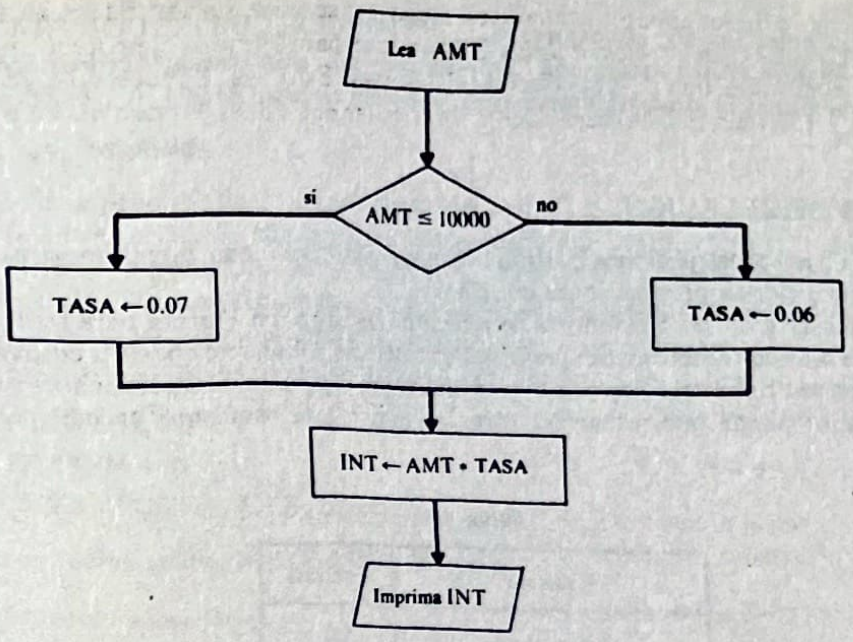


Fig. 4-5

La conexión SI... ENTONCES...DE LO CONTRARIO... supone por lo común que hay dos alternativas que pueden ocurrir, como se ilustró en los dos ejemplos anteriores. Sin embargo, hay situaciones que requieren del uso de la conexión en español SI... ENTONCES... Mostremos esto con un tercer ejemplo.

Supongamos que la EDAD de una persona y el número NDEP de sus dependientes, se perforan en una tarjeta. Queremos calcular su deducción al impuesto de renta, DEDUCT, que es de $750 por dependiente. Sin embargo, si la persona tiene 65 años de edad o más, entonces puede reclamar una deducción extra. (Suponemos que ninguno de sus dependientes tiene 65 o más años de edad.) Por esto, queremos dar al computador la orden:

SI EDAD ≥ 65 ENTONCES NDEP = NDEP + 1

antes de que calcule DEDUCT. El diagrama de flujo, para tal segmento de programa, aparece en la figura 4-6.

La proposición "IF X THEN Y" tiene el siguiente significado: cuando X es verdadero, se ejecuta Y antes de continuar con la próxima proposición; cuando X es falso, se continúa con la próxima proposición (después de saltar Y).

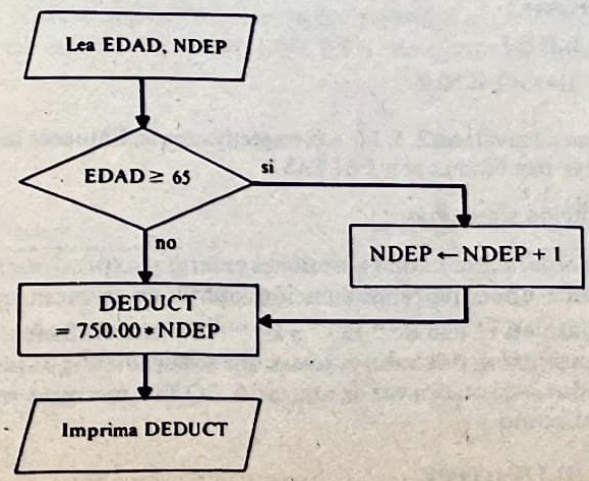


Fig. 4-6

La sutil diferencia entre las dos conexiones, SI... ENTONCES... DE LO CONTRARIO... y SI... ENTONCES..., no significa que carece de importancia. Las construcciones Fortran que pueden implementar estas conexiones se discuten en la seccion 4.5 y en el capítulo 12.

4.4 EXPRESIONES DE RELACION

Antes de introducir las construcciones Fortran sobre transferencia condicional, mostraremos primero cómo escribir en Fortran ciertas proposiciones de relación.

La tabla 4-1 da una lista de seis operadores de relación utilizados en Fortran para representar las seis relaciones matemáticas más comunes entre expresiones aritméticas. Obsérvese que cada equivalente Fortran consta de cuatro caracteres, dos letras precedidas y seguidas por un punto. Los puntos son esenciales; de lo contrario, el computador puede interpretar las letras, digamos LT, como una variable en lugar de un operador de relación.

Tabla 4-1

Relacion	Fortran
Menor que	.LT.
Menor que o igual a	.LE.
Igual	.EQ.
No igual	.NE.
Mayor que	.GT.
Mayor que o igual a	.GE.

Una *expresión de relación* consiste en dos expresiones aritméticas conectadas por *un* operador de relación. Las expresiones de relación son las formas más simples de las *expresiones lógicas*. (Las expresiones lógicas más complejas contienen también operadores lógicos los cuales se discuten en el capítulo 9.) En tiempo de ejecución, cada expresión lógica y también cada expresión de relación, representa una condición, que puede ser VERDADERA o FALSA.

EJEMPLO 4.2

Las siguientes son expresiones matemáticas y sus equivalentes expresiones de relación en Fortran.

$I < J$	I.LT.J
$(A + 3) > B$	(A + 3.0).GT.B
$5I = 2J$	5*I.EQ.2*J
$B \geq A^2$	B.GE.A**2
$(I + 3) \neq J$	(I + 3).NE.J
$(A + B^3) \leq 50$	(A + B**3).LE.50.0

Supongamos que I, J, A y B contienen los valores 2, 5, 3.0, 4.0, respectivamente. Entonces las tres primeras expresiones de relación son VERDADERAS y las tres últimas son FALSAS.

Hagamos los dos comentarios siguientes:

1. Se debe evitar la comparación entre expresiones enteras y expresiones reales, porque los valores enteros y reales tienen diferente representación interna en la memoria,
2. Se debe tener cuidado en el uso de ".EQ." y de ".NE." entre números reales, porque del error inherente en la computación, dos valores reales que se suponen iguales en teoría, pueden no serlo dentro del computador. Así que en vez de utilizar A.EQ.B, a menudo se examina su diferencia con cierta tolerancia, tal como

```
ABS(A - B).LT.0.0001
```

para compensar los posibles errores de redondeo.

4.5 PROPOSICION IF LOGICA

La transferencia condicional puede implementarse en Fortran de muchas maneras. Una de ellas es con el uso de una proposicón IF lógica, cuya forma general es como sigue:

IF(*explog*) Proposición A

Específicamente, una proposición IF lógica comienza con la palabra IF y sigue con una expresión lógica *explog* encerrada entre paréntesis. (Recordemos que la expresión lógica *explog* es una expresión que tiene el valor VERDADERO o FALSO. Véase también el capítulo 9.) Después de los paréntesis, sigue una proposición ejecutable, llamada Proposición A; sin embargo, la Proposición A no debe ser otra proposición IF ni una proposición DO (que se discutirá en el capítulo 5).

Los siguientes, son ejemplos de la proposición IF lógica:

```
IF(A.LE.B) GO TO 50
IF(L.GE.75) N = N + 1
IF(J.NE.K) WRITE(6, 30) X, Y
```

Si hablamos formalmente, la proposición IF lógica

IF(*explog*) Proposición A

tiene el siguiente significado. Si *explog* es VERDADERA, entonces se ejecuta la proposición A; de lo contrario, se salta dicha proposición.

Más precisamente, supongamos un segmento de programa Fortran que contiene

```
IF(explog) Proposición A
Proposición B
```

donde la proposición B es la *primera proposición ejecutable* que sigue a la proposición IF. Entonces, pueden ocurrir dos posibles situaciones:

1. La Proposición A no es una proposición GO TO.
2. La Proposición A es una proposición GO TO.

Caso 1

Si la Proposición A no es una proposición GO TO, entonces el anterior segmento de programa Fortran significa lo siguiente: Cuando explog es VERDADERA, primero se ejecuta la Proposición A y luego se continúa con la siguiente proposición, es decir, con la Proposición B; y cuando explog es FALSA, se continúa con la próxima proposición, o sea, con la Proposición B (saltando la Proposición A). Esta situación se muestra en la figura 4-7(*a*). Nótese que la Proposición B siempre se ejecuta. Este diagrama es equivalente al uso común de la conexión en español SI... ENTONCES... discutida en la sección 4.3.

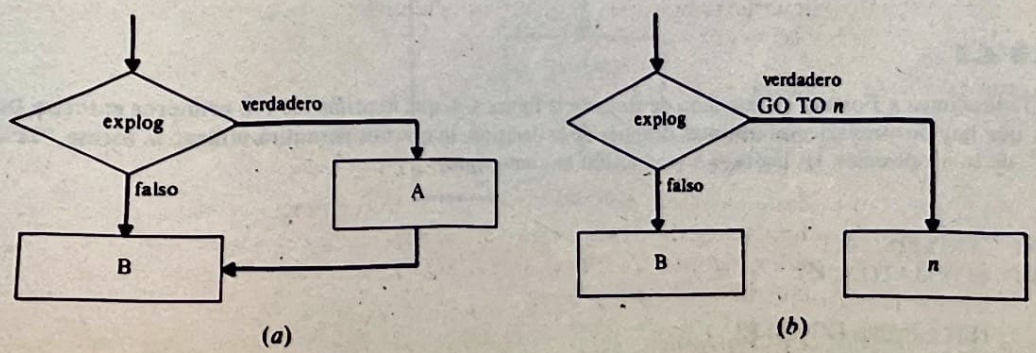


Fig. 4-7

Caso 2

Supongamos que la Proposición A es una proposición GO TO; esto es, el anterior segmento de programa Fortran es

```
IF(explog) GO TO n
Proposición B
```

Esto significa lo siguiente: Si *explog* es VERDADERA, entonces se ejecuta la parte del programa que comienza con la proposición rotulada como *n*; si *explog* es FALSA, entonces se ejecuta la porción del programa que comienza con la Proposición B. El diagrama de flujo aparece en la figura 4-7(*b*). El nos indica que las dos diferentes alternativas pueden ocurrir después de la decisión; esto es equivalente al uso común de la conexión SI... ENTONCES... DE LO CONTRARIO... De este modo, esta forma de la proposición IF lógica puede utilizarse donde quiera que una orden SI... ENTONCES... DE LO CONTRARIO... aparezca en un algoritmo. (Véase también el capítulo 12.)

Consideremos ahora el diagrama de flujo de la figura 4-5, que calcula el interés, INT, sobre un préstamo AMT. Recordemos que el diagrama de flujo provino de la orden

```
SI  AMT ≤ 10000   ENTONCES   TASA = 0.07
          DE LO CONTRARIO   TASA = 0.06
```

Supongamos que el segmento de Programa Fortran

```
SI(AMT.LE.10 000) TASA = 0.07
TASA = 0.06
```

se utiliza para implementar la orden anterior. De acuerdo con la figura 4-7(*a*) y la discusión de la página , TASA = 0.06 se ejecuta siempre. Pero esto no es lo que nosotros deseamos; por lo tanto, este segmento de programa Fortran no puede utilizarse.

A fin de no tener en cuenta TASA = 0.06 cuando AMT ≤ 10 000, debemos usar una proposición GO TO en la proposición IF lógica, como en el siguiente programa Fortran.

```
      REAL INT
      READ(5, 10) AMT
10    FORMAT(F15.2)
      IF(AMT.LE.10000.) GO TO 100
      TASA = 0.06
      GO TO 200
100   TASA = 0.07
200   INT = AMT*TASA
      WRITE(6, 20) INT
20    FORMAT(1X, 'EL INTERES ES', 2X, F10.2)
      STOP
      END
```

EJEMPLO 4.3

(*a*) Traducimos a Fortran el diagrama de flujo de la figura 4-4, que imprime los 100 primeros enteros positivos. Obsérvese que hay dos trayectorias distintas después de la decisión, lo que nos permitirá utilizar la forma "IF(*explog*) GO TO *n*" de la proposición IF lógica. La traducción es como sigue:

```
      I = 1
99    WRITE(6, 10) I
10    FORMAT(1X, I5)
      I = I + 1
      IF(I.LE.100) GO TO 99
      STOP
      END
```

Las dos proposiciones

```
IF(I.LE.100) GO TO 99
STOP
```

son la traducción Fortran de la orden.

```
SI  I ≤ 100  ENTONCES  GO TO 99 a repetir el proceso
             DE LO CONTRARIO  STOP
```

(*b*) La traducción Fortran del diagrama de flujo de la figura 4-6, que calcula la deducción al impuesto de renta de una persona, DEDUCT, es como sigue:

```
      INTEGER EDAD
      READ(5, 10) EDAD, NDEP
10    FORMAT(2I15)
      IF(EDAD.GE.65) NDEP = NDEP + 1
      DEDUCT = 750.00*FLOAT(N)
      WRITE(6, 20) DEDUCT
20    FORMAT(1X, F12.2)
      STOP
      END
```

Obsérvese la similitud de la figura 4-7(*b*) y parte de la figura 4-6, así como también, que utilizamos la función de biblioteca FLOAT para convertir el entero NDEP en un número real para evitar la mezcla de modos aritméticos.

4.6 CONTROL DE UN CICLO

Supongamos que queremos repetir un proceso (es decir, un conjunto de instrucciones), digamos 100 veces. Podemos hacer esto si se utiliza un contador I, que cuenta el número de veces que el proceso se repite. Esto es, asignamos inicialmente 1 al contador I y entonces, cada vez que el proceso se ejecuta, incrementamos el contador I en 1. Si el contador I excede a 100, terminamos la ejecución del proceso. La figura 4-8 muestra un diagrama de flujo que lleva a cabo esta tarea.

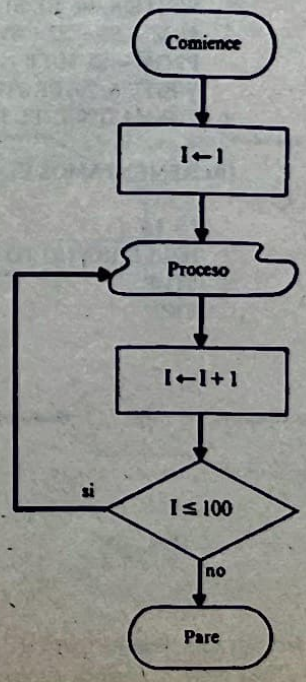


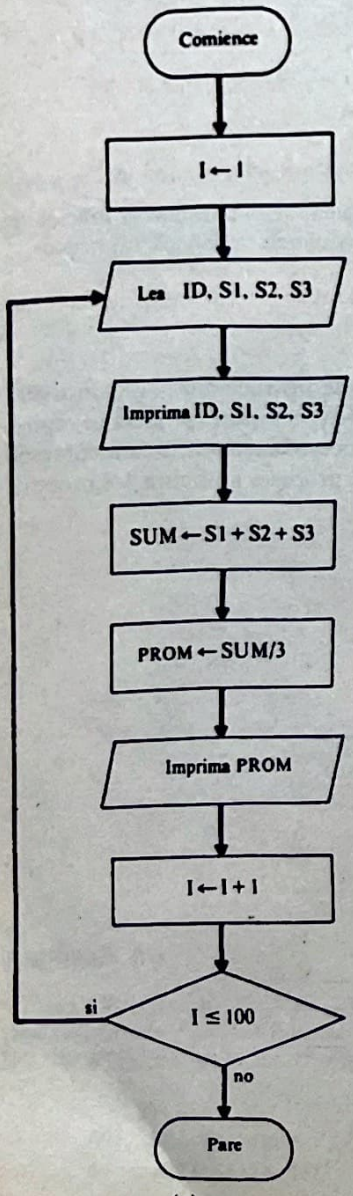
Fig. 4-8

Se advierte inmediatamente la similitud entre la figura 4-4 y la figura 4-8. En otras palabras, el mecanismo de control que utilizamos para generar los 100 primeros enteros positivos, puede también usarse como contador del proceso para controlar el final de un ciclo que se repite 100 veces.

EJEMPLO 4.4

Recordemos el ejemplo 4.1(*b*), donde encontramos el promedio de tres exámenes para cada estudiante. Escribir un programa, para el caso de que hubiese 100 tarjetas de datos.

Simplemente insertamos el diagrama de flujo del ejemplo 4.1(*b*) en el lugar apropiado en la figura 4-8 para obtener la figura 4-9(*a*). La traducción Fortran de nuestro programa se muestra en la figura 4-9(*b*). Obsérvese que la traducción Fortran de un diagrama de flujo generalmente es inmediata.



(*a*)

```
C
C     PROGRAMA PARA CALCULAR EL PROMEDIO DE TRES
C     EXAMENES
C
C
C     INICIALIZAMOS EL CONTADOR I
C
      I = 1
C
C     EL PROCESO SE REPITE
C
  100 READ(5, 10) ID, S1, S2, S3
   10 FORMAT(I4, 3F6.2)
      WRITE(6, 10) ID, S1, S2, S3
      SUM = S1 + S2 + S3
      PROM = SUM/3.0
      WRITE(6, 20) PROM
   20 FORMAT(20X, 'EL PROMEDIO ES', F6.2)
C
C     INCREMENTAMOS EL CONTADOR I
C
      I = I + 1
      IF(I.LE.100) GO TO 100
      STOP
      END
```

(*b*)

Fig. 4-9

EJEMPLO 4.5

Supongamos que deseamos escribir un programa que lea un entero N > 2 y determine si N es un número primo. Recordemos que N es un número primo si sólo es divisible por 1 o por sí mismo. Por ejemplo, 2, 3, 5, 13, 29 y 67 son números primos, pero 35 no es un número primo, puesto que es divisible por 5 y por 7. Claramente, si N no es un número primo, entonces uno de los enteros 2, 3,..., N/2 divide a N. (Compare con el problema 4.48.)

El algoritmo que tenemos en mente consta de dos partes:

1. Generamos los enteros 2, 3,..., N/2. Esto se puede hacer como se muestra en la figura 4-10.

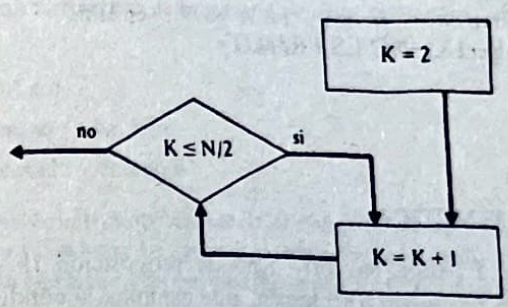


Fig. 4-10

2. Examinamos si cada uno de los enteros anteriores divide a N. Si uno de ellos lo divide, entonces N no es primo; de lo contrario, N es primo.

El diagrama de flujo de nuestro algoritmo completo aparece en la figura 4-11.

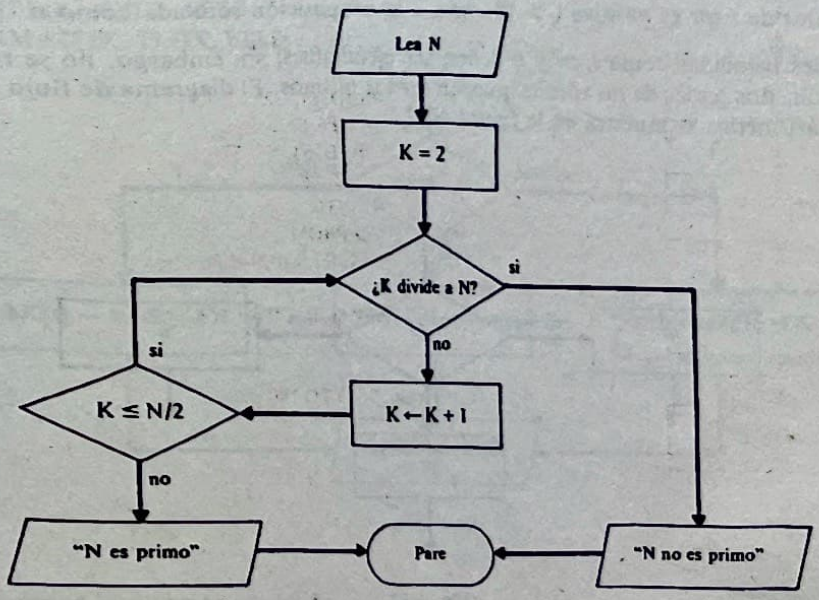


Fig. 4-11

Antes de traducir el diagrama de flujo a Fortran, tenemos que estar en capacidad de traducir "¿K divide a N?". Sin embargo, del ejemplo 2.9 sabemos que K divide a N, si y sólo si, (N/K)*K tiene el mismo valor que N. De esta manera, tenemos el programa:

```
      READ(5, 20) N
20    FORMAT(I5)
      K = 2
30    IF(N/K*K.EQ.N) GO TO 70
      K = K + 1
      IF(K.LE.N/2) GO TO 30
      WRITE(6, 40) N
40    FORMAT(1X, I5, 1X, 'ES PRIMO')
      STOP
70    WRITE(6, 50) N
50    FORMAT(1X, I5, 1X, 'NO ES PRIMO')
      STOP
      END
```

4.7 PROPOSICION IF ARITMETICA

Dentro del repertorio de Fortran, hay otro tipo de proposición IF denominada proposición IF aritmética. En contraste con la proposición IF lógica, que examina la condición lógica de una expresión lógica, una *proposición IF aritmética* examina el *signo* de una expresión aritmética. Según que ella sea positiva, cero o negativa, el control puede transferirse a varios lugares.

La forma de una proposición IF aritmética es la siguiente:

IF(*expr*) *l*, *m*, *n*

donde *expr* es una expresión aritmética y *l*, *m*, *n* son rótulos de proposiciones. La proposición nos dice:

SI el valor de *expr* es negativa (< 0), vaya a la proposición rotulada como *l*,

SI el valor de *expr* es cero ($= 0$), vaya a la proposición rotulada como *m*,

SI el valor de *expr* es positiva (> 0), vaya a la proposición rotulada como *n*.

Las proposiciones rotuladas como *l*, *m* y *n* deben ser ejecutables; sin embargo, no se requiere que sean diferentes, es decir, dos o más de los rótulos pueden ser los mismos. El diagrama de flujo equivalente de la proposicion IF aritmética se muestra en la figura 4-12.

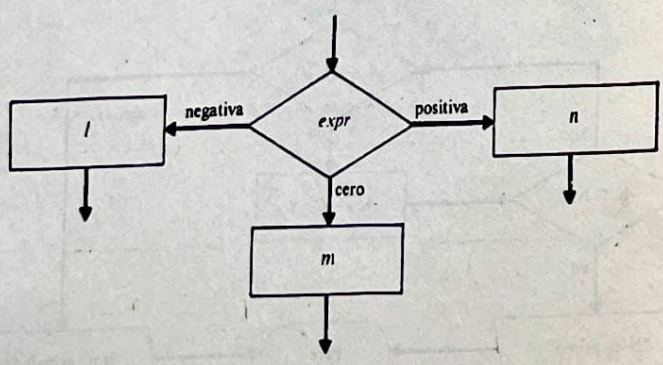


Fig. 4-12

Cualquier proposición IF aritmética puede reemplazarse de una manera, equivalente por una o más proposiciones GO TO e IF lógica. Por ejemplo,

IF(X − 2.0) 10, 20, 30 e IF(X − 2.0) 10, 10, 20

son equivalentes respectivamente a:

```
IF(X.LT.2.0) GO TO 10          IF(X.LE.2.0) GO TO 10
IF(X.EQ.2.0) GO TO 20     e    GO TO 20
GO TO 30
```

Recíprocamente, cualquier proposición IF lógica puede reemplazarse por una proposición IF aritmética equivalente. (Véase el problema 4.5.) Nótese que ciertos problemas específicos permiten en sí el uso de una proposición IF aritmética, como se ve en el siguiente ejemplo.

EJEMPLO 4.6

(*a*) Supongamos que la prima por seguro de salud se descuenta del salario de un empleado, de acuerdo con el siguiente plan:

$$\text{Prima} = \begin{cases} 9.75 & \text{si es soltero} \\ 16.25 & \text{si es casado y sin hijos} \\ 24.50 & \text{si es casado y con hijos} \end{cases}$$

Supongamos que TIPO es la variable entera que representa el código de estado civil como sigue: 1 si es soltero, 2 si es casado y sin hijos y 3 si es casado y con hijos. La figura 4-13, es un diagrama de flujo de un segmento de programa, que deduce la prima del salario de un empleado, suponiendo que se dan PAGO, TIPO e ID (identificación del empleado.) Adviértase que TIPO − 2 es negativa, cero o positiva, según que TIPO sea igual a 1, 2 ó 3. La traducción Fortran de la figura 4-13 es como sigue:

```
      NETO REAL
      IF(TIPO − 2) 10, 20, 30
10  NETO= PAGO − 9.75
      GO TO 75
20  NETO= PAGO − 16.25
      GO TO 75
    NETO =PAGO − 24.50
75    WRITE(6, 40) ID, NETO
40    FORMAT(1X, I5, 3X, F12.2)
```

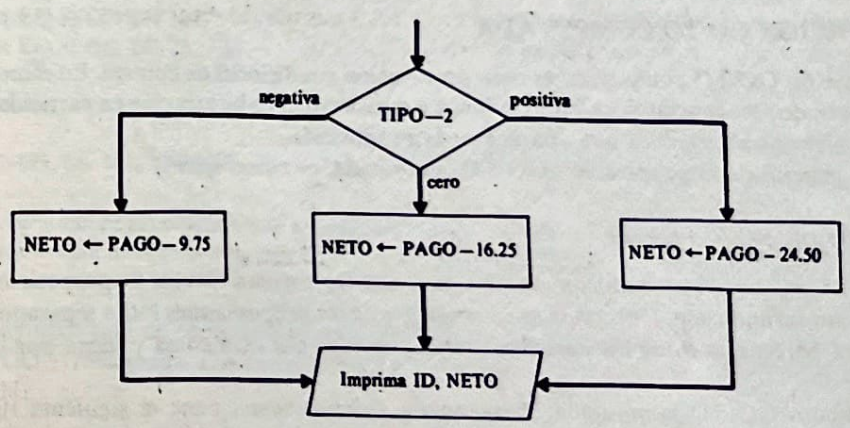


Fig. 4-13

(*b*) Consideremos la ecuación cuadrática

$$ax^2 + bx + c = 0$$

donde a, b, y c son números reales y $a \neq 0$. Existirán raíces reales solamente cuando el discriminante $D = b^2 - 4ac$ sea no negativo. Las raíces están dadas por

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Obsérvese que si el discriminante $D = 0$, entonces obtenemos la raíz múltiple $-b/2a$. Escribamos un programa Fortran que calcule las raíces reales de un conjunto de coeficientes A, B y C.

```
C
C     CALCULE LAS RAICES REALES DE UNA ECUACION CUADRATICA DONDE A
C       NO ES CERO
C
      READ(5, 11) A, B, C
   11 FORMAT(3F10.2)
      WRITE(6, 12) A, B, C
   12 FORMAT('0', 'LOS COEFICIENTES SON', 3(2X, F10.2))
      D = B**2 - 4.0*A*C
C
C     EXAMEN DEL DISCRIMINANTE
C
      IF(D) 22, 33, 44
   22 WRITE(6, 13)
   13 FORMAT(1X, 'NO HAY RAICES REALES')
      GO TO 10
   33 RAIZ = -B/2.0*A
      WRITE(6, 14) RAIZ, RAIZ
   14 FORMAT(1X, 'HAY DOS RAICES IDENTICAS', 2(3X, F10.2))
      GO TO 10
   44 RAIZ1 = (-B + SQRT(D))/(2.0*A)
      RAIZ2 = (-B - SQRT(D))/(2.0*A)
      WRITE(6, 15) RAIZ1, RAIZ2
   15 FORMAT(1X, 'HAY DOS RAICES REALES DISTINTAS', 2(3X, F10.2))
   10 STOP
      END
```

4.8 PROPOSICION GO TO COMPUTADA

La proposición GO TO computada es otra proposición condicional de control. En efecto, ella permite cualquier número de escogencias para bifurcación. La decisión de una bifurcación en particular, se basa en el valor de cualquier variable entera que aparece en la proposición.

La forma general de la proposición GO TO computada, es como sigue:

GO TO $(n_1, n_2, \ldots, n_k)$, J

Donde, $n_1, n_2, \ldots, n_k$ son constantes enteras sin signo, que representan rótulos de proposiciones y J es una variable entera no subindicada. Obsérvese que los rótulos de las proposiciones están separados por comas y encerrados entre paréntesis y que los paréntesis están seguidos por otra coma y luego por la variable llamada J.

La proposición GO TO computada, mencionada anteriormente, tiene el siguiente significado: En tiempo de ejecución, primero se examina el valor de la variable entera J. Si el valor de J es 1, el control se transfiere a la proposición rotulada como n_1 (es decir, se ejecuta la proposición rotulada como n_1); si el valor de J es 2, entonces se ejecuta la proposición rotulada como n_2 y así, sucesivamente. Es extremadamente importante que, en tiempo de ejecución, el valor de la variable J esté entre 1 y k; de lo contrario, no se puede predecir la consecuencia. En efecto, algunos compiladores ni siquiera dan un mensaje de error.

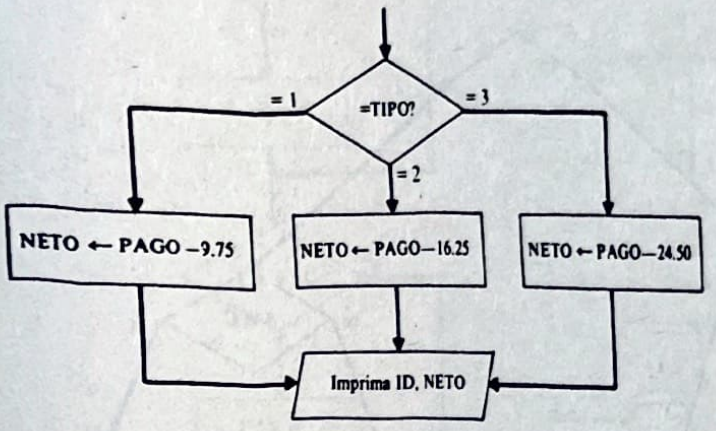
Supongamos, por ejemplo, que se ejecuta la siguiente proposición:

GO TO (10, 15, 70, 22, 15), NEW

Entonces, el control se transfiere a la proposición rotulada como 10, 15, 70, 22 ó 15, según que NEW tome los valores 1, 2, 3, 4 ó 5, respectivamente. Obsérvese que los rótulos de las proposiciones no necesitan ser distintos. Enfatizamos que en tiempo de ejecución, NEW deberá ser positiva y no podrá exceder de 5.

EJEMPLO 4.7

Consideremos el problema del ejemplo 4.4, donde la prima por seguro de salud se deduce de conformidad con que el empleado sea soltero (TIPO 1), casado y sin hijos (TIPO 2), o casado y con hijos (TIPO 3). El diagrama de flujo y el segmento de programa Fortran que utiliza una proposición GO TO computada, aparece en la figura 4-14.




```
      GO TO (10, 20, 30), TIPO
10    NETO  PAGO-9.75
      GO TO 75
20    NETO=PAGO−16.25
      GO TO 75
30    NETO=PAGO−24.50
75    WRITE(6, 40) ID, NETO
40    FORMAT(1X, I5, 3X, F12.2)
```

Fig. 4-14

Se ve fácilmente que la proposición GO TO computada no nos proporciona una capacidad extra en la solución de problemas. Esto es, podemos siempre arreglar una secuencia lógica de proposiciones IF lógica que llevan a cabo el mismo resultado que una sola proposición GO TO computada. Por ejemplo,

```
IF(MM.EQ.1) GO TO 99
IF(MM.EQ.2) GO TO 88
IF(MM.EQ.3) GO TO 101
IF(MM.EQ.4) GO TO 23
```

es equivalente a

```
GO TO (99, 88, 101, 23), MM
```

Sin embargo, un conjunto de proposiciones IF lógica es menos eficiente que una sola proposición GO TO computada, la que solamente requiere un paso para transferir el control al punto deseado. Como puede verse, por lo común, requiere menos escritura.

Nota. Hay otro tipo de proposición condicional de control que permite múltiples bifurcaciones y es la proposición GO TO asignada. Discutiremos esta proposición y la proposición ASSIGN que la acompaña en el capítulo 11.

4.9 ALGORITMOS

Discutiremos dos programas en esta sección. El propósito es mostrar los diferentes pasos que encierra la escritura de un programa, desde el comienzo de un algoritmo, hasta su refinamiento y finalmente hasta su implementación. La traducción a Fortran de los diagramas de flujo finales no se hace aquí, pero se deja como ejercicio (problemas suplementarios) al lector.

Supongamos que nos dan un paquete de cuatro tarjetas, en cada una de las cuales se perfora un número. Deseamos encontrar el mayor de estos cuatro números. Como solamente hay cuatro números, los leemos de la memoria y los llamamos A, B, C y D. El siguiente algoritmo nos da el número mayor suponiendo que A, B, C y D están en la memoria:

1. Compare A y B y observe cuál es mayor.
2. Compare C con el resultado de 1 y observe cuál es mayor.
3. Compare D con el resultado de 2 y observe cuál es mayor.

La implementación intuitiva del algoritmo anterior se ilustra en el diagrama de flujo de la figura 4-15.

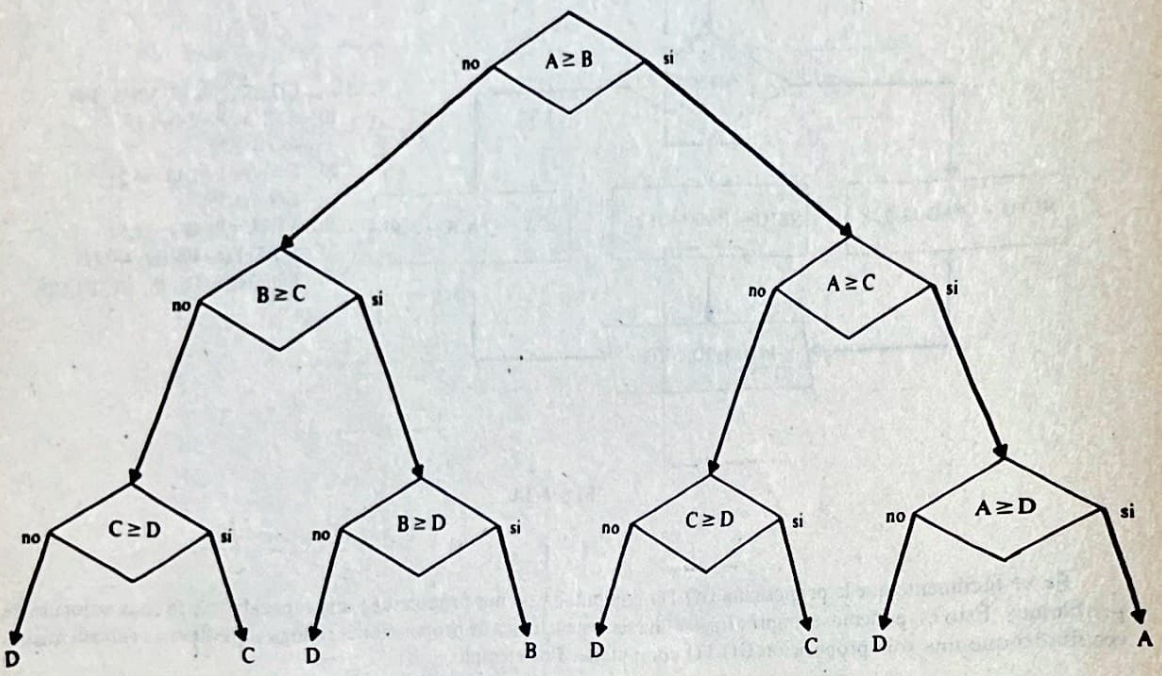


Fig. 4-15

Obsérvese que hay $7 = 1 + 2 + 2^2$ cajas de decisión en la figura 4-15. En efecto, el número total de cajas de decisión en dicho algoritmo aumenta exponencialmente con el número de elementos. Veamos: si hubiese cinco números, habría $2^3 + 7 = 15$ cajas de decisión; si hubiese seis números, habría $2^4 + 15 = 31$ cajas de decisión y así, sucesivamente. Obviamente, este algoritmo no es satisfactorio cuando el número de elementos es muy grande.

Refinemos un poco nuestro algoritmo permitiendo que LAR denote el actual valor mayor y utilicemos los cuatro siguientes pasos:

(0) Asigne A a LAR; esto es, sea LAR = A.
(1) Compare LAR y B, y guarde el mayor en LAR.
(2) Compare LAR y C, y guarde el mayor en LAR.
(3) Compare LAR y D, y guarde el mayor en LAR.

El diagrama de flujo de este algoritmo se da en la figura 4-16(*a*). Obsérvese que el número de cajas de decisión es tres, y que aumenta linealmente con el número de elementos. Esto es, para cinco números, hay cuatro cajas de decisión y para k números, se tendrán $k-1$ cajas de decisión. Claramente, este algoritmo es mejor en relación con el algoritmo precedente.

Una de las cosas más sorprendentes acerca del diagrama de flujo de la figura 4-16(*a*) es que un cierto grupo de cajas y de flujo de información se repite una y otra vez. Este grupo se muestra en la figura 4-16(*b*) donde ? representa un elemento arbitrario (diferente del primero). Por lo tanto, si podemos asegurar que el valor de ? cambia correctamente, entonces podemos encerrar las cajas en un ciclo. Este cambio se puede llevar a cabo, si se leen los valores individualmente y si se les asigna el mismo nombre (localización de almacenamiento) cada vez. Esto produce el ciclo infinito exhibido en la figura 4-16(*c*).

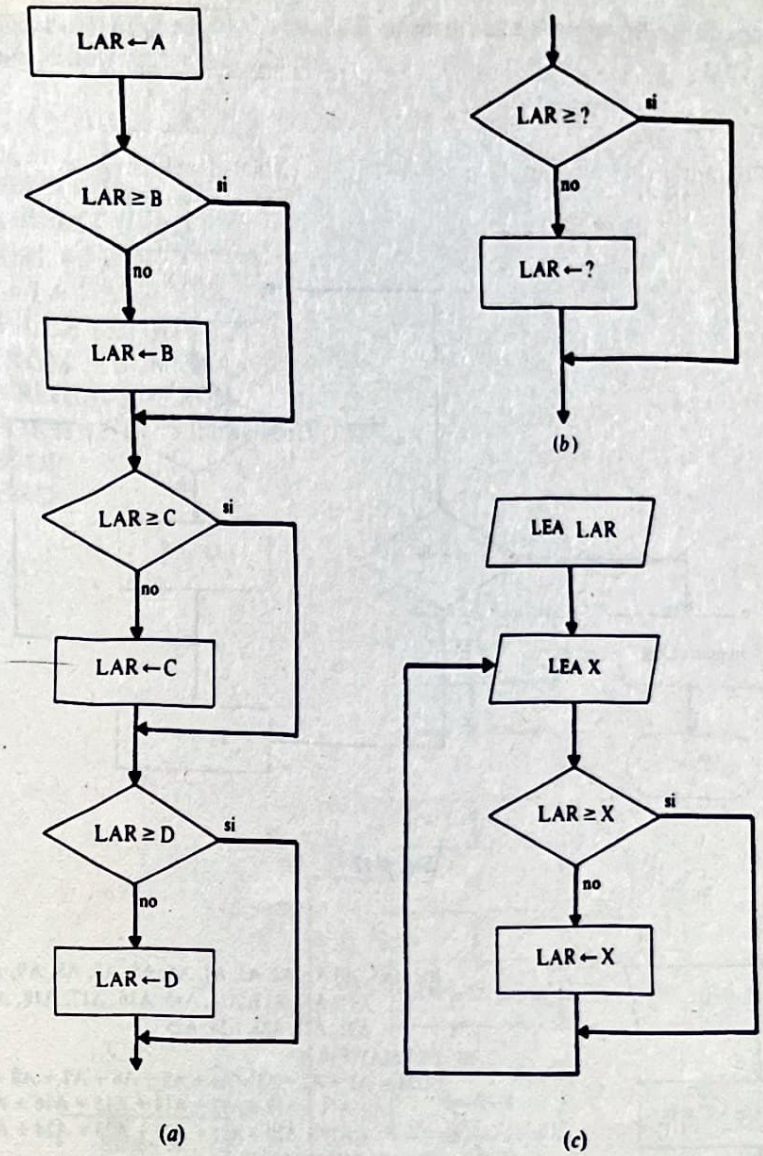


Fig. 4-16

Ahora supongamos que hay 100 tarjetas en un paquete dado. El diagrama de flujo para encontrar el mayor número aparece en la figura 4-17. (Puesto que LAR se asigna inicialmente al primer número, el proceso se repite solamente 100 – 1 = 99 veces.) El programa Fortran para un problema un poco más general aparece en el problema 4.13.

Consideremos ahora otro ejemplo. Supongamos que nos dan un paquete de cuatro tarjetas en cada una de las cuales se perfora un número. Deseamos encontrar el promedio (media aritmética) de estos números. La manera más obvia es leer estos números en localizaciones, digamos, A, B, C y D y entonces dividir su suma por 4. El diagrama de flujo aparece en la figura 4-18(*a*).

Ahora, supongamos que el número de tarjetas en el paquete es de 25 en vez de 4. Si se utiliza el esquema anterior, se requeriría la lectura de 25 variables distintas, digamos A1, A2,..., A25. (Los numerales 1, 2,..., 25 en los nombres de las variables no cumplen otro papel que representar símbolos diferentes.) La figura 4-18(*b*) da el programa Fortran que calcula el promedio de 25 números. Obsérvese que las proposiciones READ y SUM son muy extensas y que claramente están propensas a error. Además, si hubiese 100 números, este esquema sería dispendioso y debería reescribirse el programa. Por lo tanto, sería valioso tener un esquema general (algoritmo) tal, que el mismo programa (posiblemente con modificaciones menores) pueda utilizarse para calcular la suma de 4 ó 25 ó 100 números. Discutiremos esto enseguida.

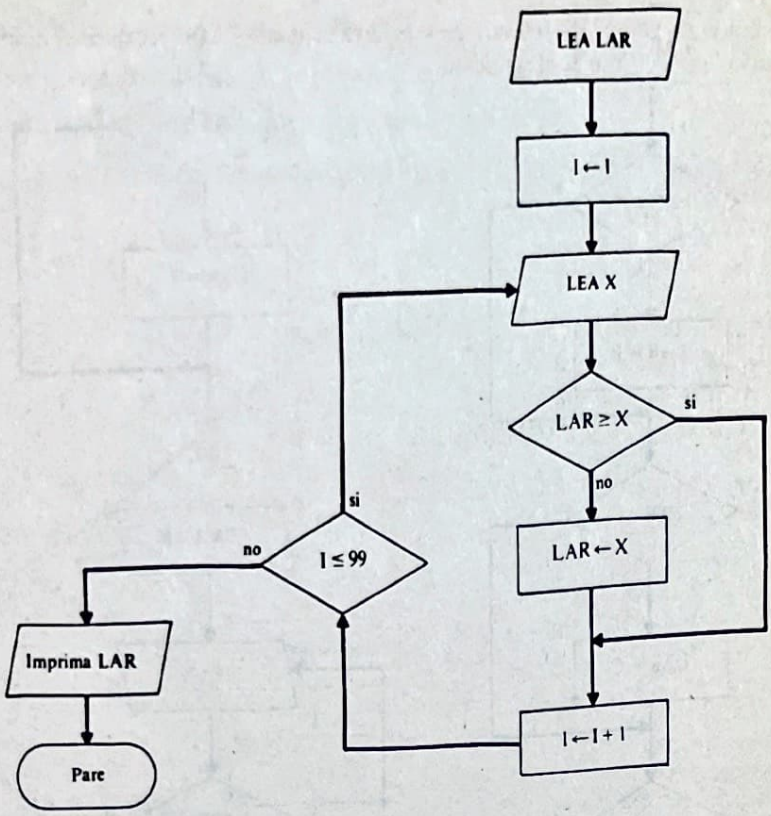
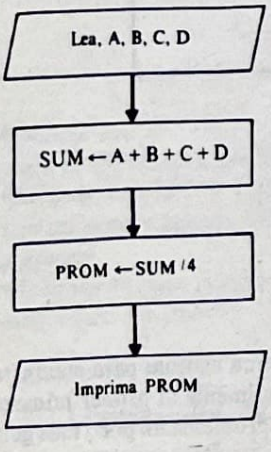


Fig. 4-17



(*a*)

```
      READ(5, 10) A1, A2, A3, A4, A5, A6, A7, A8, A9, A10,
     1        A11, A12, A13, A14, A15, A16, A17, A18, A19, A20,
     1        A21, A22, A23, A24, A25
   10 FORMAT(F10.1)
      SUM = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 + A9 + A10
     1        + A11 + A12 + A13 + A14 + A15 + A16 + A17 + A18
     1        + A19 + A20 + A21 + A22 + A23 + A24 + A25
      PROM= SUM/25.0
      WRITE(6, 20) PROM
   20 FORMAT(1X, 'EL PROMEDIO DE LA CLASE ES', 1X, F4.1)
      STOP
      END
```

(*b*)

Fig. 4-18

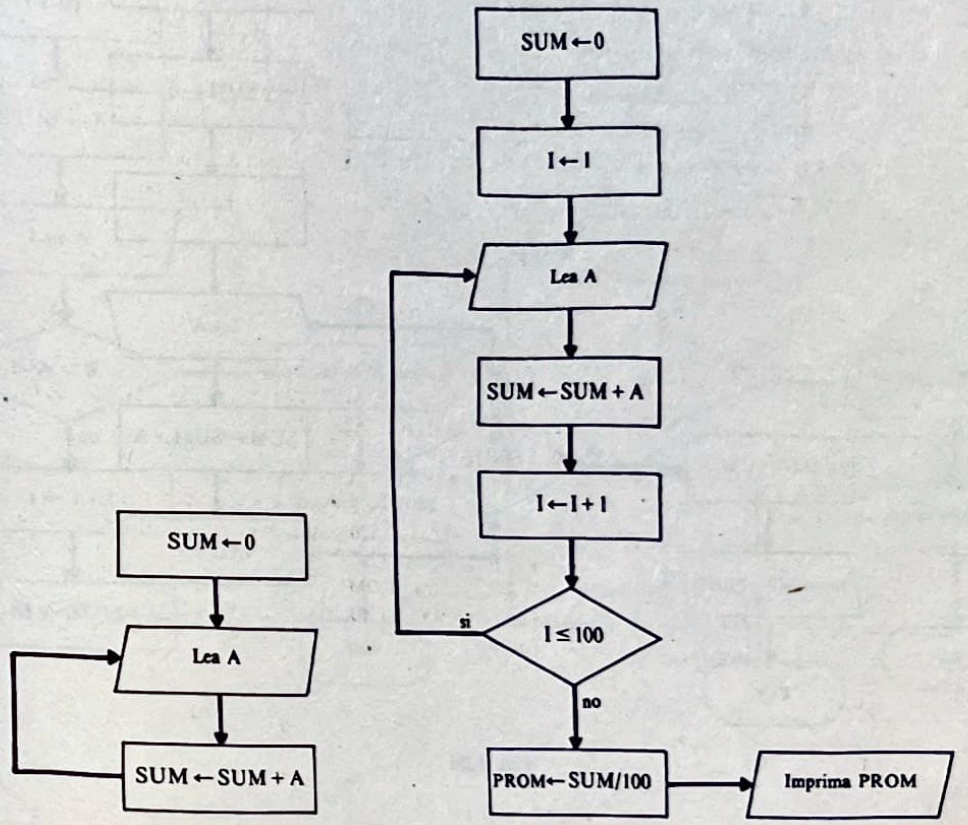
Para calcular A1 + A2 + A3 + A4 + .., podemos considerarlo como

((((A1) + A2) + A3) + A4) + · · ·

Esto es, se puede comenzar con SUM = 0 y luego añadir estos números a SUM uno cada vez; en efecto, pueden sumarse uno a uno cada vez que se leen de la memoria. Esto se ilustra en la figura 4-19(*a*) en un ciclo infinito. Ahora, si hay 100 tarjetas en el paquete con un número perforado en cada una de ellas, se puede establecer un contador I para contar el número de veces que se ejecuta el ciclo. Podemos entonces parar el

proceso de adición una vez que los 100 números se hayan sumado. Esto produce el diagrama de la figura 4-19(*b*). Su equivalente Fortran es el siguiente:

```
      SUM = 0.0
      I = 1
   50 READ(5, 10) A
   10 FORMAT (F10.2)
      SUM = SUM + A
      I = I + 1
      IF(I.LE.100) GO TO 50
      PROM = SUM/100.0
      WRITE(6, 20) PROM
   20 FORMAT(1X, 'EL PROMEDIO ES', 1X, F4.1)
      STOP
      END
```



(*a*) (*b*)

Fig. 4-19

4.10 TARJETAS DE ENCABEZAMIENTO Y DE DELIMITACION DE DATOS

Los programas desarrollados en la sección 4.9 tienen una desventaja, especialmente porque sólo funcionan apropiadamente si hay exactamente 100 tarjetas en el paquete de datos. Si un paquete dado consta de 76 tarjetas, tenemos que modificar el programa, aunque se utilice el mismo algoritmo. Claramente, lo que

deseamos es escribir un programa que sea independiente del número de tarjetas en el paquete de datos, es decir, que funcione para cualquier número de datos, sea 4 ó 100 ó 76. Discutiremos dos métodos por medio de los cuales esto se lleva a cabo: (1) el uso de una tarjeta de encabezamiento y (2) el uso de una tarjeta de delimitación.

En el diagrama de flujo de la figura 4-19(*b*), para encontrar la suma de 100 valores, el ciclo se repite 100 veces. Supongamos ahora que deseamos que el ciclo se repita un número variable de veces (dependiendo del problema particular). Debemos identificar este número por un nombre de variable, digamos N. Para ejecutar el programa, se reemplaza el valor 100 por la variable N, cuyo valor debe definirse en tiempo de ejecución. Esto se puede lograr si se perfora en una tarjeta, denominada *tarjeta de encabezamiento*, el número de tarjetas que componen el paquete; esta tarjeta se coloca al comienzo del paquete de datos. Definimos, entonces, N con una instrucción de lectura. El diagrama de flujo de este programa se da en la figura 4-20.

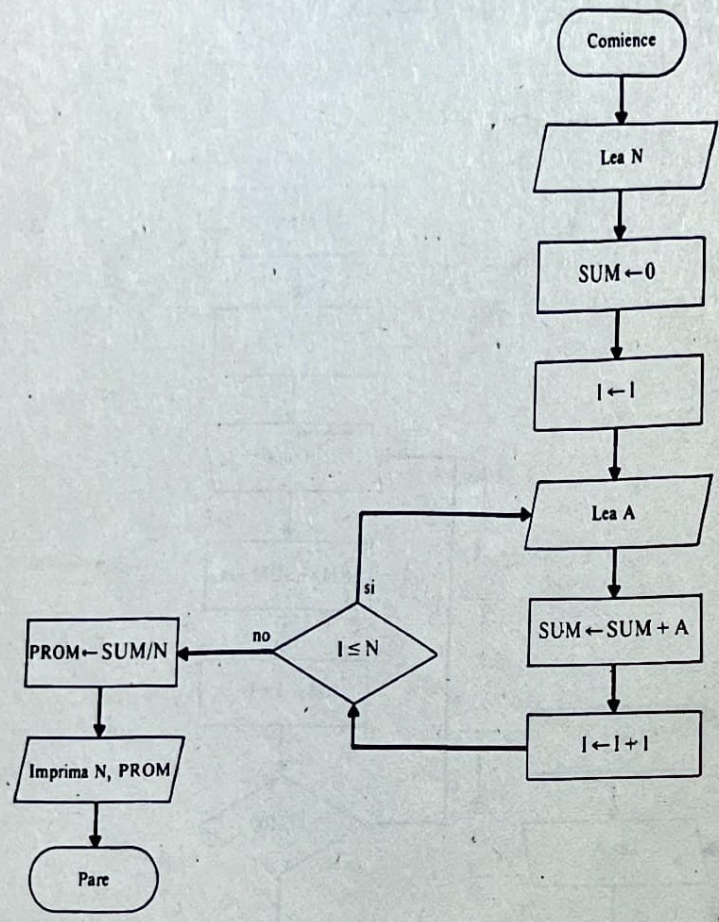


Fig. 4-20

Algunas veces, el uso de una tarjeta de encabezamiento es casi imposible. Por ejemplo, supongamos que realmente calculamos el promedio de las notas correspondientes a un examen de un curso de introducción a la programación de computadores, que quizá tiene 200-300 estudiantes. A fin de utilizar una tarjeta de encabezamiento, alguien debe contar el número de tarjetas de datos en el paquete. Puesto que hay 200-300 estudiantes, se tendrán 200-300 tarjetas. Naturalmente, sería preferible que el computador ejecutara el conteo. Esto se puede conseguir con el uso de una tarjeta de delimitación, que vamos a discutir enseguida.

Supongamos que vamos a calcular el promedio de las notas de un examen como el anterior. Puesto que una nota está representada por un número solamente si él está comprendido entre 0 y 100, podemos colocar, al final del paquete, una tarjeta adicional que contenga un número que esté fuera de este intervalo, por ejemplo, un número negativo. Esta tarjeta se denomina una *tarjeta de delimitación*. Cuando el computador encuentra un número negativo, señala el fin del paquete de datos. El diagrama de flujo de dicho programa aparece en la figura 4-21(*a*).

Nota. Obsérvese que en este programa, el contador se inicializa en cero. La razón de esto es que solamente deseamos conocer la cantidad de números positivos y no sabemos que un número es positivo, sino hasta cuando es examinado. En este caso, el valor de I después de I = I + 1 indica cuál fue la última tarjeta leída y no cuál es la que se está leyendo.

La técnica anterior opera solamente cuando conocemos el intervalo de los números. Sin embargo, se puede utilizar una modificación de esta técnica cuando no se tiene idea acerca de los datos que se van a manipular. Se puede leer un valor adicional, digamos K, y perforar un número, digamos 1, en el campo para K en la tarjeta de delimitación. No es necesario alterar las tarjetas del paquete de datos, puesto que los blancos se leen como ceros (en los campos numéricos). Por lo tanto, el computador detecta el fin de los datos cuando K tenga el valor 1. El diagrama de flujo de este programa alterno aparece en la figura 4-21(*b*).

Por último, señalemos que algunas veces las tarjetas de encabezamiento y de delimitación de datos se denominan *tarjetas centinelas*.

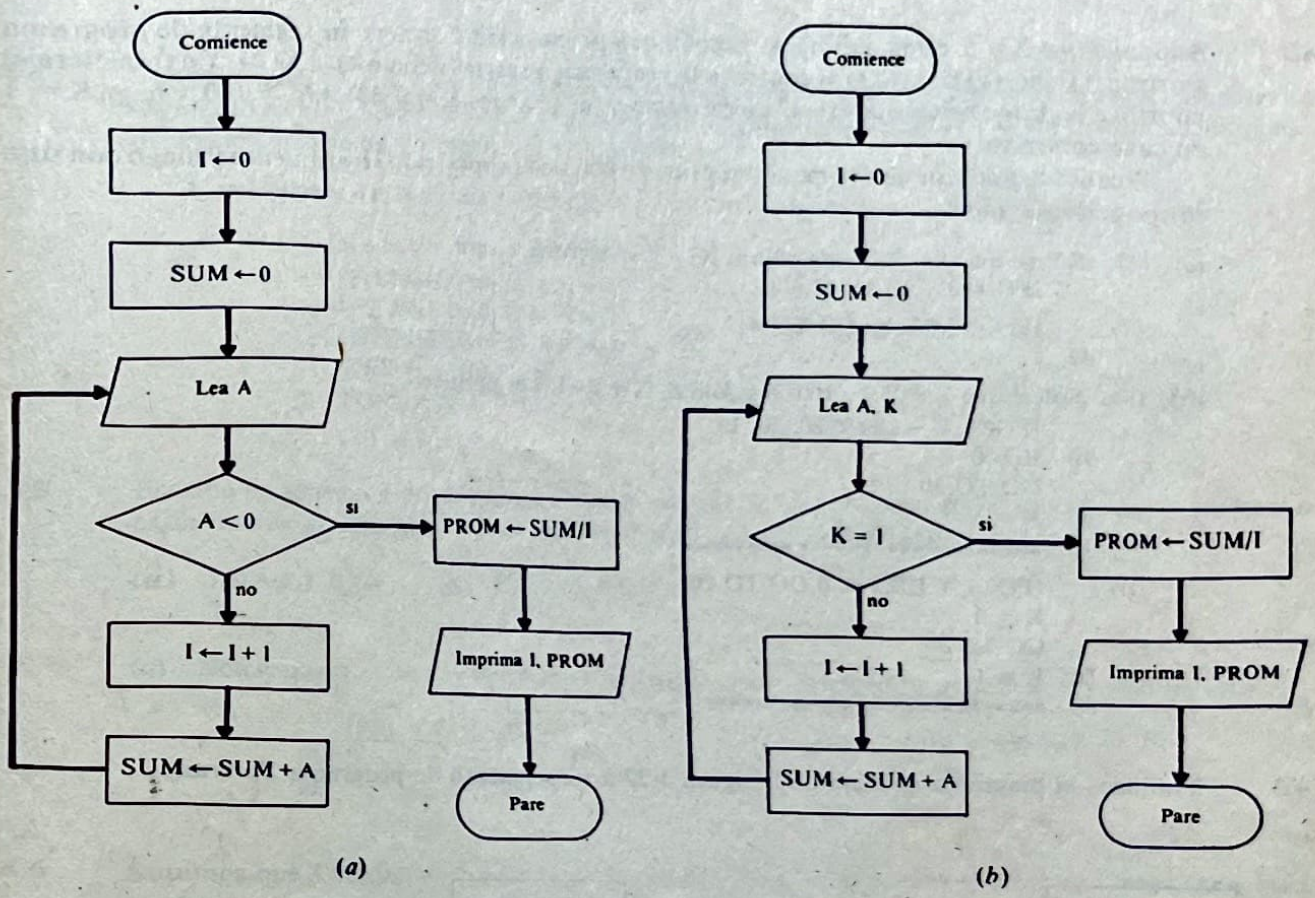


Fig. 4-21

Problemas resueltos

PROPOSICIONES IF

4.1 Escriba las siguientes proposiciones en Fortran:

(*a*) Si X > Y, pare.
(*b*) Si J ≠ K, vaya a la proposición rotulada como 31.
(*c*) Si $A^2 \leq B + C$, vaya a la proposición rotulada como 41.
(*d*) Si $A-B \geq X^3$, pare.

```
(a) IF(X.GT.Y) STOP
(b) IF(J.NE.K) GO TO 31
(c) IF(A**2.LE.B + C) GO TO 41
(d) IF(A - B.GE.X**3) STOP
```

4.2 Suponga que X y Y están definidos. Escriba una proposición Fortran o un segmento de programa Fortran, tal que (*a*) transfiera el control a la proposición rotulada como 41 si $X^2 \leq Y$ o transfiera el control a la proposición número 42 en caso contrario; (*b*) haga K = 0 si X + Y > 100, y haga K = 1 en caso contrario.

Realice lo anterior de dos maneras, primero con una proposición IF aritmética y luego con una proposición IF lógica.

(*a*) (*i*) Nótese que $X^2 \leq Y$, si y sólo si $X^2 - Y$ es negativa o cero.

```
      IF(X**2 - Y) 41, 41, 42
```

(*ii*)

```
      IF(X**2.LE.Y) GO TO 41
   42 .....................
```

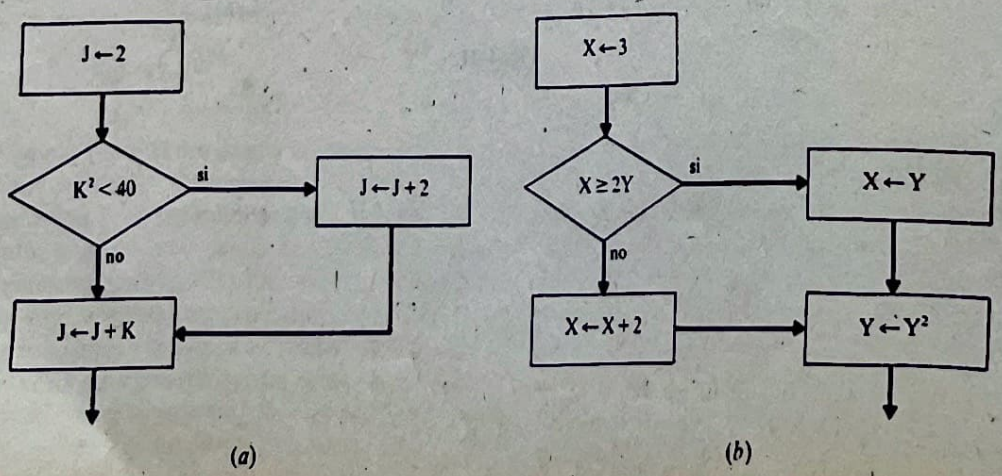
(*b*) (*i*) Nótese que X + Y > 100, si y sólo si, X + Y−100 es positivo.

```
      IF(X + Y - 100.0) 20, 20, 10
   10 K = 0
      GO TO 30
   20 K = 1
   30 ****************************
```

(*ii*)

```
      IF(X + Y.LE.100.0) GO TO 20
      K = 0
      GO TO 30
   20 K = 1
   30 ****************************
```

4.3 Traduzca el diagrama de flujo de la figura 4-22 a un segmento de programa Fortran.



(*a*) (*b*)

Fig. 4-22

(*a*) Esto es equivalente a SI .. ENTONCES... discutido en el Caso (1) de la sección 4.5.

```
      J = 2
      IF(K**2.LT.40) J = J + 2
      J = J + K
```

(*b*) Esto es equivalente a SI... ENTONCES... DE LO CONTRARIO... discutido en el Caso (2) de la sección 4.5.

```
      X = 3.0
      IF(X.GE.2*Y) GO TO 10
      X = X + 2.0
      GO TO 20
10    X = Y
20    Y = Y**2
```

4.4 Suponga que T1, T2 y T3 están en la memoria y denotan resultados de exámenes. Escriba un segmento de programa Fortran que encuentre e imprima el número de estos resultados ≥ 90. (En el programa 4.14 se da un problema más general.)

Consideremos que N denota el número de resultados ≥ 90. (Por lo tanto, N debe ser igual a 0, 1, 2 ó 3.)

```
      N = 0
      IF(T1.GE.90.0) N = N + 1
      IF(T2.GE.90.0) N = N + 1
      IF(T3.GE.90.0) N = N + 1
      WRITE(6, 10) N
10    FORMAT(1X, I5)
```

4.5 Escriba un segmento de programa Fortran que utilice una proposición IF aritmética equivalente a cada una de las siguientes proposiciones (aquí S y T son proposiciones ejecutables.)

(*a*)
```
IF(A.LT.B) S
T
```

(*b*)
```
IF(J.NE.K) S
T
```

(*a*) Nótese que A < B, si y sólo si A−B es negativa.

```
      IF(A - B) 10, 20, 20
10    S
20    T
```

(*b*) Nótese que $J \neq K$, si y sólo si $J-K \neq 0$.

```
      IF(J - K) 10, 20, 10
10    S
20    T
```

4.6 Suponga que J contiene el valor 5 y K contiene el valor 10. Encuentre el valor de J después de cada segmento de programa:

(*a*)
```
IF(3*J.LT.K) J = J + 2
J = J + 3
```

(*b*)
```
IF(2*J.EQ.K) J = J + 2
J = J + 3
```

(*c*)
```
      IF(K - J) 10, 20, 10
10    J = K
20    J = J + 1
```

(*d*)
```
      IF(4*J - 2*K) 10, 20, 20
10    J = K
20    J = J + 1
```

(*e*)
```
      IF(2*K.LE.3*J) GO TO 50
      J = J + 1
      GO TO 60
50    J = K
60    J = J + K
```

(*f*)
```
      IF(K.GT.J) GO TO 50
      J = J + 1
      GO TO 60
50    J = K
60    J = J + K
```

(*a*) Puesto que 3J < K no es cierto, solamente se ejecuta J = J + 3; por lo tanto, el valor final de J es 8.
(*b*) Como 2J = K es verdadero, primero se ejecuta J = J + 2, lo cual da J = 7. Después se ejecuta J = J + 3; por lo tanto, el valor final de J es 10.
(*c*) Puesto que K – J es positiva, primero se ejecuta J = K, lo cual da J = 10. Luego se ejecuta J = J + 1, lo que da un valor final de J = 11.
(*d*) Como 4J – 2K es cero, solamente se ejecuta J = J + 1. Por lo tanto, el valor final de J es 6.
(*e*) Puesto que 2K ≤ 3J es falso, primero se ejecuta J = J + 1, lo que da como resultado J = 6. Después se ejecuta J = J + K, lo que da un resultado final de J = 16.
(*f*) Como K > J es verdadero, primero se ejecuta J = K, lo cual da J = 10. Luego se ejecuta J = J + K, lo que finalmente da J = 20.

PROPOSICION GO TO COMPUTADA

4.7 Encuentre los errores, si los hay, en cada una de las proposiciones GO TO computada:

(*a*) GO TO (5, 8, 4) MARK
(*b*) GO TO (5, 22, 22, 57), KKK
(*c*) GO TO (2, 84, 578), ERIK
(*d*) GO TO (5, 76, 0, 24), J

(*a*) Debe haber una coma antes de MARK.
(*b*) No hay errores.
(*c*) ERIK debe ser una variable entera.
(*d*) El cero no puede ser un rótulo de proposición.

4.8 Encuentre el rótulo de la proposición al cual se transfiere el control en cada uno de los siguientes segmentos de programa:

(*a*)
```
MARK = 3
GO TO (23, 47, 16, 94), MARK
```

(*b*)
```
J = 2
J = J + 2
GO TO (23, 47, 16, 94), J
```

(*c*)
```
K = 1
K = K + 3
GO TO (23, 16, 94), K
```

(*a*) Puesto que MARK = 3, el control se transfiere a la proposición rotulada como 16, la del tercer número en la lista.
(*b*) J = 4 cuando se ejecuta la proposición GO TO computada; así que el control se transfiere a la proposición rotulada como 94.
(*c*) K = 4 cuando se ejecuta la proposición GO TO computada. Sin embargo, solamente hay tres números en la lista; por lo tanto, se dará un mensaje de error.

PROGRAMAS

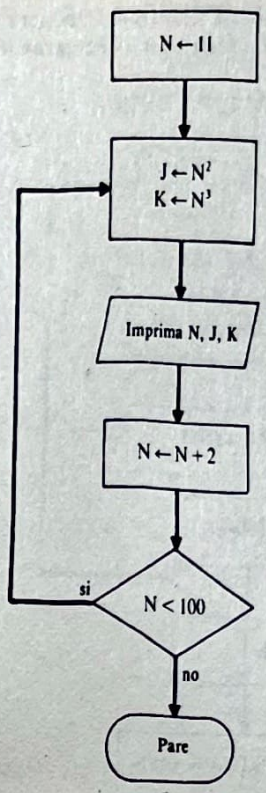
4.9 Dibuje un diagrama de flujo y escriba un programa que imprima un número impar N, su cuadrado N^2 y su cubo tal, que los diferentes N's aparezcan en líneas diferentes. El programa deberá escribirse de manera que la columna de los N's se encabece por NUMERO, la columna de N^2's por CUADRADO y la columna de N^3's por CUBO.

Consideremos N = 11, J = N^2 y K = N^3. Después de incrementar N en 2, examinamos si N < 100. El diagrama de flujo se da en la figura 4-23(*a*) y la traducción Fortran en la figura 4-23(*b*).

Si añadimos las dos siguientes proposiciones al comienzo del programa,

```
    WRITE(6, 5)
5   FORMAT(7X,'NUMERO', 3X, 'CUADRADO', 5X, 'CUBO')
```

entonces, la salida es como se muestra en la figura 4-23(*c*).



(*a*)

```
      N = 11
10    J = N**2
      K = N**3
      WRITE(6, 20) N, J, K
20    FORMAT(1X, 3I10)
      N = N + 2
      IF(N.LT.100) GO TO 10
      STOP
      END
```

(*b*)

NUMERO	CUADRADO	CUBO
11	121	1331
13	169	2197
15	225	3375
.....	.....	.....
97	9409	912673
99	9801	970299

(*c*)

Fig. 4-23

4.10 Dibuje un diagrama de flujo y escriba un programa Fortran para calcular (con cinco lugares decimales):

$$(a)\ \frac{1}{1}+\frac{1}{3}+\frac{1}{5}+\cdots+\frac{1}{21}, \qquad (b)\ \frac{2}{1}\cdot\frac{4}{3}\cdot\frac{6}{5}\cdot\cdots\cdot\frac{22}{21}$$

(*a*) Consideramos primero SUM = 0 y luego añadimos sucesivamente 1/K a SUM para K = 1, 3, 5, ..., 21. La figura 4-24(*a*) da el diagrama de flujo. Nótese que consideramos primero K = 1 y luego incrementamos K en 2 cada vez, para que la suma solamente incluya los números impares. Además, examinamos si K ≤ 21, puesto que no deseamos añadir algo después de la suma de 1/21. En la traducción del diagrama de flujo a Fortran tenemos que escribir

```
1.0/FLOAT(K)
```

para 1/K puesto que no queremos realizar división con enteros. El programa es el siguiente:

```
      SUM = 0.0
      K = 1
11    SUM = SUM + 1.0/FLOAT(K)
      K = K + 2
      IF(K.LE.21) GO TO 11
      WRITE(6, 20) SUM
20    FORMAT(1X, 'LA SUMA ES', 2X, F8.5)
      STOP
      END
```

(*b*) Consideramos inicialmente K = 1 y luego multiplicamos sucesivamente PRDT por (K + 1)/K para K = 1, 3,..., 21. La figura 4-24 da el diagrama de flujo. Nótese la similitud con la parte (*a*). El programa es el que sigue:

```
      PRDT = 1.0
      K = 1
   13 PRDT = PRDT*FLOAT(K + 1)/FLOAT(K)
      K = K + 2
      IF(K.LE.21) GO TO 13
      WRITE(6, 30) PRDT
   30 FORMAT(1X, 'EL PRODUCTO ES', 2X, F8.5)
      STOP
      END
```

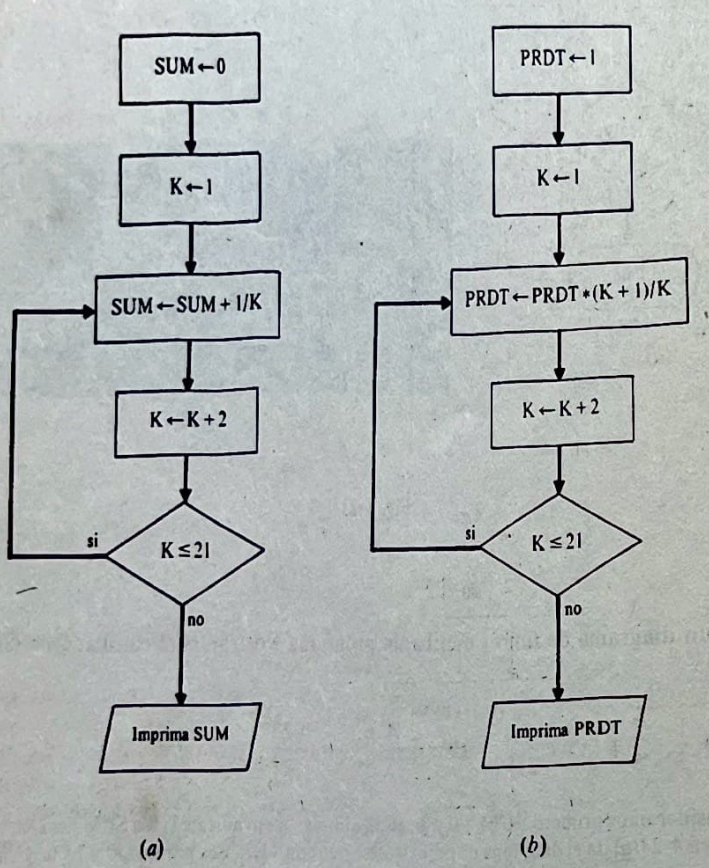


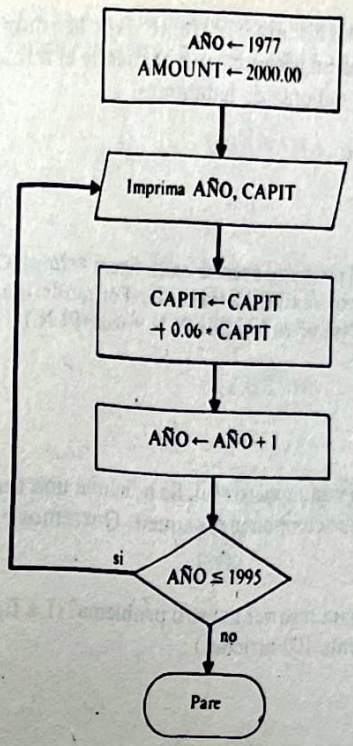
Fig. 4-24

4.11 Supongamos que se deposita un capital de $2000.00 en una cuenta de ahorros en 1977 y supongamos, también que el banco paga anualmente 6 por ciento de interés compuesto. Escriba un programa que imprima el AÑO y el CAPIT de la cuenta hasta 1995.

Cada año, CAPIT se incrementa en un 6%. De esta manera la asignación

CAPIT ← AMOUNT + 0.06*CAPIT

se repite mientras AÑO ≤ 1995. El diagrama de flujo para el programa y la traducción Fortran aparecen en la figura 4-25.

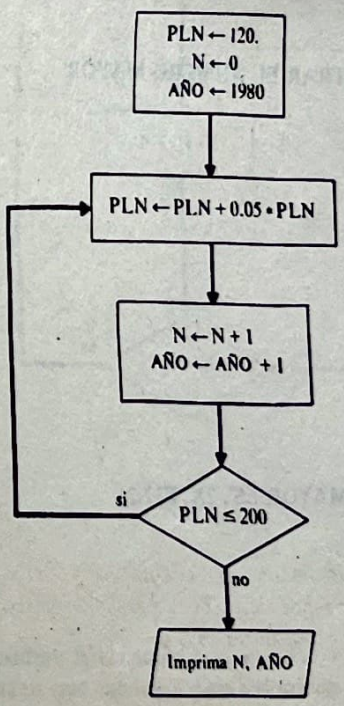


(*a*)

```
      INTEGER AÑO
      WRITE(6, 10)
10    FORMAT('1', 6X, 'AÑO', 6X, 'CAPIT'//)
      AÑO = 1977
      CAPIT = 2000.00
55    WRITE(6, 20) AÑO , CAPIT
20    FORMAT(1X, I10, 3X, '$', F8.2)
      CAPIT = CAPIT + 0.06*CAPIT
      AÑO = AÑO + 1
      IF(AÑO.LE.1995) GO TO 55
      STOP
      END
```

(*b*)

Fig. 4-25



(*a*)

```
      INTEGER  AÑO
      PLN = 120.0
      N = 0
      AÑO = 1980
100   PLN = PLN + 0.05*PLN
      N = N + 1
      AÑO = AÑO + 1
      IF(PLN.LE.200.) GO TO 100
      WRITE(6, 10) N, AÑO
10    FORMAT(1X, I3, 3X, I4)
      STOP
      END
```

(*b*)

Fig. 4-26

4.12 Supongamos que en 1980 la población (PLN) de un país alcanza a 120 millones de habitantes y que cada año la población aumenta a la rata de 5%. Escriba un programa que encuentre el número N de años y el AÑO en el cual la población excede a 200 millones de habitantes.

Cada año PLN aumentó 5%. De este modo, la asignación

$$PLN \leftarrow PLN + 0.05 * PLN$$

se repite mientras PLN $\leq$ 200. El diagrama de flujo para el programa aparece en la figura 4-26(*a*). Obsérvese que imprimimos N y AÑO solamente después de que PLN excede a 200. La traducción Fortran del diagrama de flujo aparece en la figura 4-26(*b*). (Se podría escribir PLN*1.05 en lugar de PLN + 0.05*PLN.)

4.13 Supongamos que cada tarjeta de un paquete contiene un número real. Se adiciona una tarjeta de encabezamiento que contiene el número de tarjetas que componen el paquete. Queremos encontrar el número mayor en el paquete.

(*a*) ¿Qué cambios deben hacerse en la figura 4-17 para resolver nuestro problema? (La figura 4-17 se refiere a un paquete que contiene exactamente 100 tarjetas.)

(*b*) Escriba el programa.

(*a*) Necesitamos solamente insertar "Lea N" al comienzo del diagrama de flujo, y cambiar I $\leq$ 99 por I $\leq$ N − 1. De otra manera, el diagrama de flujo sería idéntico.

(*b*) El programa aparece enseguida. Obsérvese que necesitamos una proposición de declaración de tipo para que LAR sea una variable REAL.

```
C
C       PROGRAMA PARA ENCONTRAR EL NUMERO MAYOR
C
        REAL LAR
        READ(5, 10) N
   10   FORMAT(I6)
        READ(5, 20) LAR
        I = 1
  100   READ(5, 20) X
   20   FORMAT(F12.2)
        IF(LAR.GE.X) GO TO 200
        LAR = X
  200   I = I + 1
        IF(I.LE.N - 1) GO TO 100
        WRITE(6, 30) LAR
   30   FORMAT(1X, 'EL NUMERO MAYOR ES', 2X, F12.2)
        STOP
        END
```

4.14 Suponga que cada tarjeta de un paquete contiene el resultado de un examen de un estudiante. El paquete tiene una tarjeta de delimitación en la que se ha perforado un número negativo. Dibuje un diagrama de flujo y escriba el programa Fortran que cuente el número I de estudiantes que tomaron el examen y que dé el número N de exámenes $\geq$ 90.

La figura 4-27 es el diagrama de flujo. Nótese que I y N se inicializan primero en cero. La traducción Fortran es la siguiente:

```
C
C       PROGRAMA DE CALIFICACIONES
C
        I = 0
        N = 0
  100   READ(5, 10) X
   10   FORMAT(F10.1)
        IF(X.LT.0.0) GO TO 200
        I = I + 1
        IF(X.GE.90.0) N = N + 1
        GO TO 100
  200   WRITE(6, 20) I, N
   20   FORMAT('0', I10, 2X, 'ESTUDIANTES QUE TOMARON EL EXAMEN'/
     1      '0', I10, 2X 'ALCANZARON 90')
        STOP
        END
```

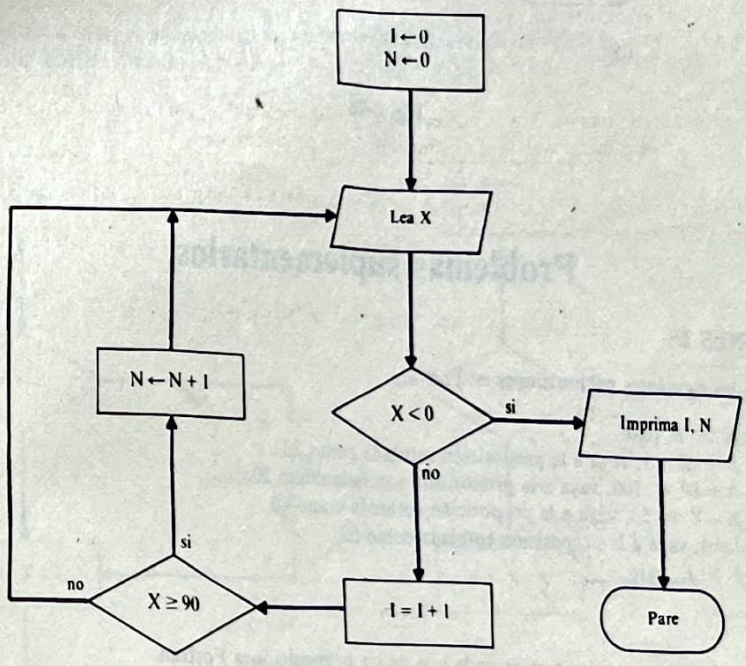
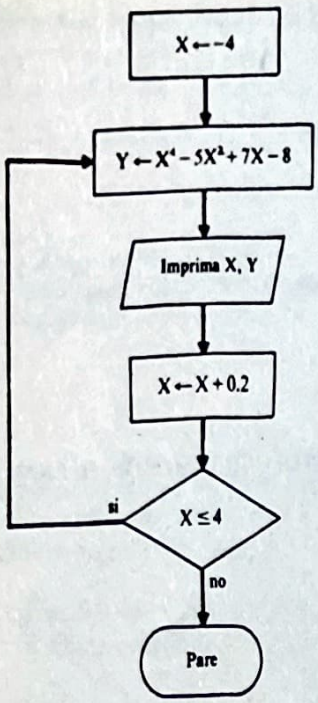


Fig. 4-27

4.15 Considere la ecuación $y = x^4 - 5x^2 + 7x - 8$. Dibuje un diagrama de flujo y escriba un programa Fortran que calcule y para valores de x desde -4 hasta 4 con incrementos de 0.2. Imprima x y su correspondiente valor y en diferentes líneas.

Primero asignamos -4 a X. Después, calculamos Y e imprimimos X, Y; luego, incrementamos X en 0.2. Repetimos el proceso mientras $X \leq 4$. El diagrama de flujo del algoritmo aparece en la figura 4-28(*a*); su traducción Fortran se muestra en la figura 4-28(*b*).



(*a*)

```
      X = -4.0
100   Y = X**4 - 5.0*X**2 + 7.0*X - 8.0
      WRITE(6, 10) X, Y
10    FORMAT(6X, 'X =', F5.1, 3X, 'Y =', 2X, F6.2)
      X = X + 0.2
      IF(X.LE.4.0) GO TO 100
      STOP
      END
```

(*b*)

Fig. 4-28

Problemas suplementarios

PROPOSICIONES IF

4.16 Escriba las siguientes proposiciones en Fortran:

(*a*) Si A > B, pare.
(*b*) Si J = K + 3, vaya a la proposición rotulada como 20.
(*c*) Si A + B² < 100, vaya a la proposición rotulada como 30.
(*d*) Si X − Y ≥ 50, vaya a la proposición rotulada como 40.
(*e*) Si I ≠ 4, vaya a la proposición rotulada como 50.
(*f*) Si I ≤ K, pare.

4.17 Encuentre los errores, si los hay, en cada una de las proposiciones Fortran:

(*a*) IF(A = B) GO TO 50
(*b*) IF(X GT Y) STOP
(*c*) IF(B**2 − A*C) STOP
(*d*) IF(X.LT.Y + Z) 10, 15, 20
(*e*) IF(X.LE.100) GO TO K
(*f*) IF(A − 100) 10, 20, 30,
(*g*) IF(X.GE.Y) GO TO 55
(*h*) IF(INTEREST.LT.AMOUNT STOP

4.18 Realice cada uno de los siguientes enunciados de dos maneras, primero con proposiciones IF lógica y luego con proposiciones IF aritmética:

(*a*) Transfiera el control a la proposición número 100 si A + B² > 100 ó transfiera el control en caso contrario a la proposición rotulada como 200.
(*b*) Haga K = 1 si A − B ≤ 100 o haga K = 2, en caso contrario.
(*c*) Transfiera el control a las proposiciones rotuladas como 10, 20 ó 30, según que J < K, J = K, J > K.

4.19 Reescriba las siguientes proposiciones utilizando proposiciones IF aritmética. (Aquí S y T son proposiciones ejecutables.)

(*a*) IF(X.GT.Y) S
T

(*b*) IF(I.EQ.J) S
T

(*c*) IF(X.LE.Y) S
T

(*d*) IF(A.GE.B) S
T

4.20 Suponga que J y K contienen los valores 3 y 5 respectivamente. Encuentre el valor final de J después de cada segmento de programa:

(*a*)
```
IF(J.GE.K) J = J + 2
J = J + 2
```

(*b*)
```
IF(5*J.EQ.3*K) J = J + 2
J = J + 2
```

(*c*)
```
      IF(J - K) 10, 10, 20
10    J = K
20    J = J + 2
```

(*d*)
```
      IF(J.LT.K - 1) GO TO 10
      J = J + 2
10    J = J + K
```

(*e*)
```
      IF(J.GE.K + 1) GO TO 10
      J = J + 2
10    J = J + K
```

(*f*)
```
      IF(2*J - K) 10, 10, 20
10    J = K
20    J = J + 2
```

4.21 Traduzca a un segmento de programa Fortran cada diagrama de flujo de la figura 4-29. En cada caso, encuentre el valor final de Z si: (*a*) A = 2.0, B = 3.0; (*b*) A = 3.0, B = 2.0.

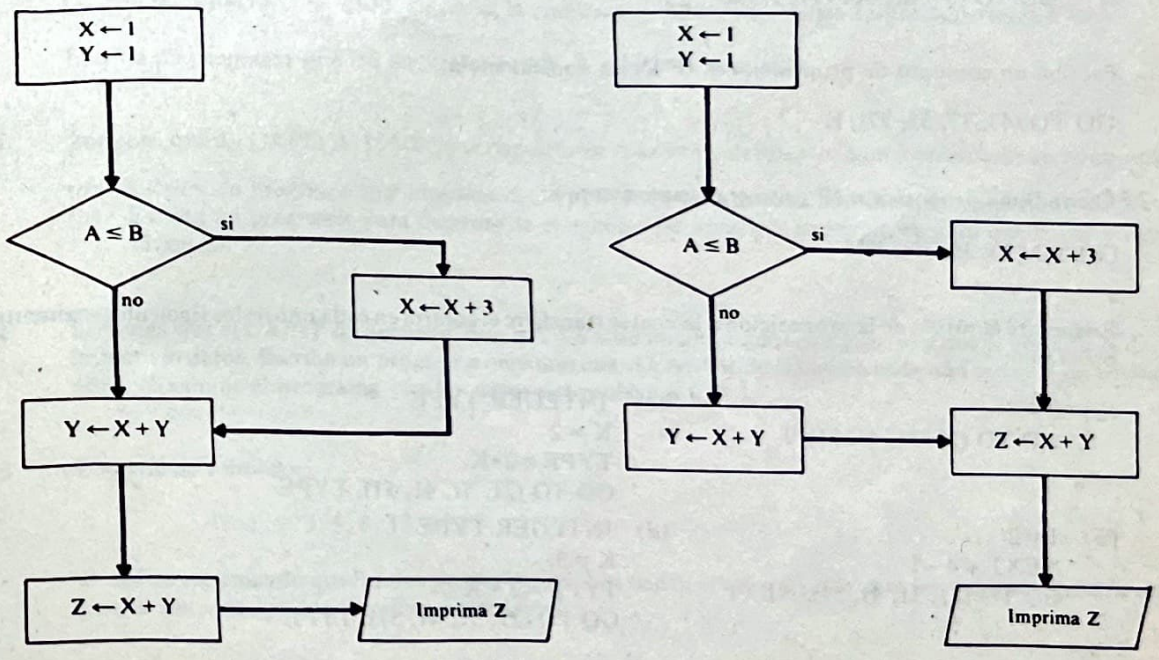


Fig. 4-29

4.22 Traduzca a un segmento de programa Fortran el diagrama de flujo de la figura 4-30. Encuentre el valor de Z, si:

(*a*) A = 2.0, B = 3.0, C = 3.0, D = 2.0
(*b*) A = 3.0, B = 2.0, C = 3.0, D = 2.0
(*c*) A = 3.0, B = 2.0, C = 2.0, D = 3.0
(*d*) A = 2.0, B = 3.0, C = 2.0, D = 3.0

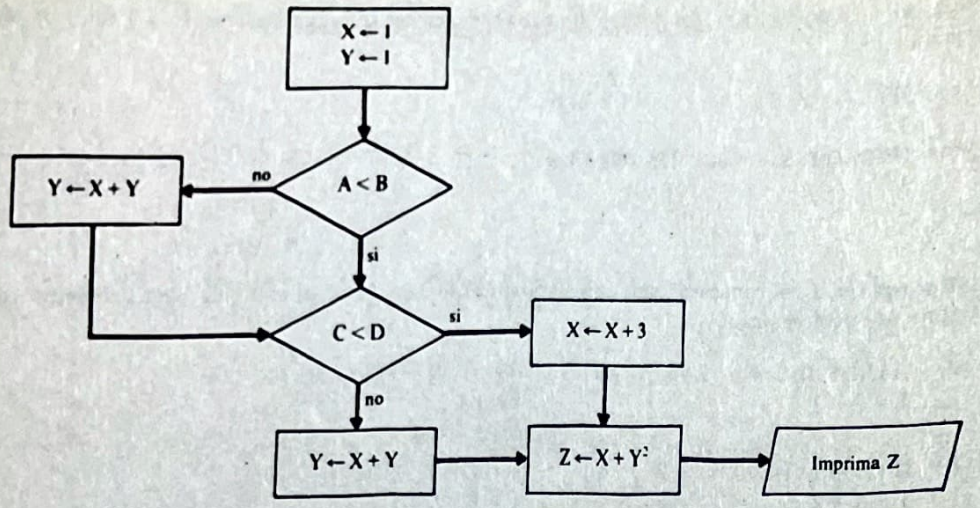


Fig. 4-30

PROPOSICION GO TO COMPUTADA

4.23 Encuentre los errores, si los hay, en cada una de las proposiciones GO TO computada:

(*a*) GO TO (35, 17, 17, 46) JIM
(*b*) GO TO (42, 0, 88, 1234), LAST
(*c*) GO TO (34, 34, 58, 58, 34), N237K
(*d*) GO TO (234, 2345678, 7654), J
(*e*) GO TO (34, 82, 56, 77), TOM

4.24 Escriba un conjunto de proposiciones IF lógica equivalente a:

```
GO TO (47, 33, 55, 77), K
```

4.25 Escriba una proposición IF aritmética equivalente a:

```
GO TO (20, 30, 40), JIM
```

4.26 Encuentre el rótulo de la proposición a la cual se transfiere el control en cada uno de los siguientes segmentos de programa:

(*a*)
```
J = 3
GO TO (21, 31, 41, 51), J
```

(*b*)
```
I = 2
NEXT = 4 - I
GO TO (21, 31, 41, 51), NEXT
```

(*c*)
```
INTEGER TYPE
K = 2
TYPE = 2*K
GO TO (21, 31, 41, 51), TYPE
```

(*d*)
```
INTEGER TYPE
K = 3
TYPE = 2 + K
GO TO (21, 31, 41, 51), TYPE
```

PROGRAMAS

4.27 Escriba un programa que imprima los enteros positivos desde 1 hasta 300, de manera que en la salida aparezcan tres números en una línea, así:

```
  1    2    3
  4    5    6
 ..................
298  299  300
```

4.28 Escriba un programa que calcule, con dos lugares decimales, la suma

$$1/2 + 2/3 + 3/4 + \cdots + 99/100$$

4.29 Escriba un programa que lea un entero positivo $N \geq 10$ y calcule, con dos lugares decimales, las sumas

(*a*) $1 + 1/2 + 1/3 + \cdots + 1/N$
(*b*) $1 - 1/2 + 1/3 - \cdots \pm 1/N$

Supongamos que A está definido. Calcule también:

(*c*) $$\frac{1}{1+A} + \frac{1}{1+2A} + \frac{1}{1+3A} + \cdots + \frac{1}{1+N \cdot A}$$

4.30 Escriba un programa que lea un entero positivo $N \geq 10$ y calcule, con cinco lugares decimales, el producto

$$\frac{1}{1^2} \cdot \frac{3}{2^2} \cdot \frac{5}{3^2} \cdot \cdots \cdot \frac{2N-1}{N^2}$$

4.31 Escriba un programa Fortran para calcular el pago BRUTO, si ID (Identificación de los empleados), TASA (prehora) y las HORAS (número de horas trabajadas por semana) se almacenan todas en la memoria; por cada hora o fracción, se paga tiempo extra (después de 40 horas).

4.32 La comisión sobre las VENTAS totales de un empleado es como sigue:

(*a*) Si VENTAS < \$50, entonces no hay comisión.
(*b*) Si \$50 ≤ VENTAS ≤ \$550, entonces comisión = 10% de las VENTAS
(*c*) Si las VENTAS > \$500, entonces la comisión = \$50 + 8% de las ventas superiores a \$500.

Escriba un programa que lea las ventas totales y calcule la comisión.

4.33 Suponga que un CAPIT de \$1 000.00 se deposita en una cuenta de ahorros a un interés compuesto anual del 7%.

(*a*) Escriba un programa que imprima el CAPIT de la cuenta cada año durante un período de 20 años.
(*b*) Escriba un programa para determinar el número de años que se requieren para que la cuenta exceda a \$5 000.00.

4.34 Suponga que el CAPIT de depósito, que la TASA de interés anual (compuesto) y que el número N de años se leen como datos. Escriba un programa que imprima el VALOR de la cuenta cada año durante un período de N años. (Examine el programa con los datos del problema 4.33.)

4.35 La serie de Fibonacci

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

se define suponiendo que los dos primeros términos tienen el valor 1 y que cada término subsiguiente está dado por la suma de los dos términos precedentes:

$$1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8, \ldots$$

(*a*) Escriba un programa que imprima la serie de Fibonacci, de manera que no exceda de 10 000.
(*b*) Escriba un programa que imprima los 50 primeros términos de la serie de Fibonacci.

4.36 Consideremos que J y K son enteros positivos con J ≤ K. La serie generalizada de Fibonacci se define suponiendo que J es el primer término, K el segundo término y que cada término subsiguiente es la suma de los dos términos anteriores. Escriba un programa que lea J y K e imprima los 50 primeros términos de la serie generalizada de Fibonacci.

4.37 Cada tarjeta de un paquete contiene un número real positivo (menor que 1000), si el paquete tiene una tarjeta de delimitación.

(*a*) Escriba un programa que encuentre el número mayor en el paquete. (Compare con el problema 4.13, el cual utiliza una tarjeta de encabezamiento.)
(*b*) Escriba un programa que encuentre el número menor en el paquete.
(*c*) Escriba un programa que encuentre el segundo número mayor en el paquete. (Suponga que el paquete tiene más de un número.)

4.38 Traduzca a Fortran la figura 4-19(*b*), que encuentra la media aritmética de 100 números.

4.39 Traduzca a Fortran la figura 4-21(*a*), que encuentra la media aritmética de un conjunto de números no negativos. El paquete tiene una tarjeta de delimitación en la que se perfora un número negativo.

4.40 Suponga que cada tarjeta de un paquete contiene el resultado como una cifra entera de un examen de un estudiante. (Se añade una tarjeta de delimitación, la cual contiene un número negativo.) A un resultado R de un examen se asigna A, B, C, D ó E, según que $R \geq 90, 80 \leq R < 90, 70 \leq R < 80, 60 \leq R < 70$ ó $R < 60$. Escriba un programa que encuentre:

(*a*) El número I de estudiantes que tomaron el examen.
(*b*) El número de A's, B's, C's, D's y F's.
(*c*) El número de exámenes perfectos, es decir, el número K de 100's. (Esto generaliza el problema 4.14.)

4.41 Suponga que, actualmente, las poblaciones de los países A y B son 52 y 85 millones de habitantes, respectivamente. Supongamos que las tasas de crecimiento de población son 6% y 4%, respectivamente. Escriba un programa que imprima las poblaciones (a la unidad de mil más cercana) de A y B cada año hasta cuando la población de A exceda la de B y que encuentre el número N de años para el cual la población de A excede a la de B.

4.42 Repita el problema 4.41, pero ahora las poblaciones y tasas de crecimiento de los países A y B, digamos POBLA y POBLB y TASAA y TASAB, respectivamente, se dan en tarjetas de datos. (Examine el programa con los datos del problema 4.41.)

4.43 (*a*) Escriba un programa que imprima todos los enteros positivos impares que sean menores que 100, sin incluir los enteros divisibles por 7:

1, 3, 5, 9, 11, 15, 17, 19, 23, . . . , 97, 99

(*b*) Reescriba el programa anterior de manera que aparezcan cuatro números en una línea, es decir, que la salida sea como

```
 1    3    5    9
11   13   17   19
23   25   27   29
. . . . . .
```

4.44 Escriba un programa que lea un entero positivo N, donde $10 \leq N \leq 1000$ y que imprima en una nueva página, el número N y todos sus divisores, de manera que la salida aparezca como sigue (si N es digamos 12):

```
DIVISORES DE 12
   1
   2
   3
   4
   6
  12
```

4.45 Para cada estudiante de un curso de educación continuada se prepara una tarjeta que indica la edad del estudiante, el sexo, el puntaje en la clase y el estado civil. Estos datos se dan de acuerdo con los códigos:

(*i*) 1 si es mujer, 2 si es hombre.
(*ii*) 1 si está en kinder, 2 si está en primaria, 3 si está en bachillerato, 4 si está en la universidad, 5 si está en post-grado y 6 si no está matriculado.
(*iii*) 1 si es soltero, 2 si es casado

Se utiliza una tarjeta de delimitación para terminar el paquete. Escriba un programa que calcule el porcentaje de: (*a*) hombres, (*b*) mujeres, (*c*) estudiantes de post-grado, (*d*) estudiantes con puntaje superior a 30.

4.46 Para los datos del problema 4.45 escriba un programa que calcule los porcentajes de: (*a*) estudiantes de post-grado casados, (*b*) mujeres con puntaje superior a sus compañeros (bachillerato y universidad), (*c*) número de estudiantes de 1 a 4 con puntaje superior a 30.

4.47 En un paquete, cada tarjeta contiene tres números positivos A, B y C. El paquete tiene una tarjeta de delimitación.

(*a*) Escribir un programa que determine si A, B y C pueden formar los lados de un triángulo. Si lo forman, calcule el perímetro del triángulo; si no, imprima el mensaje 'NO FORMAN UN TRIANGULO'. (Sugerencia: A, B y C pueden formar un triángulo si cada lado es menor que la suma de los otros dos, es decir, si $A < B+C$, $B < A+C$ y $C < A+B$.)
(*b*) Escriba un programa que determine si A, B, C forman los lados de un: (*i*) triángulo equilátero (tres lados iguales), (*ii*) triángulo isósceles (dos lados iguales), (*iii*) triángulo rectángulo (hipotenusa^2 = lado^2 + lado^2).

4.48 El ejemplo 4.5 da un programa que determina si un entero $N > 2$ es un número primo. Esto se hace examinando si N es divisible por cualquiera de los enteros 2, 3,..., N/2. Si N no es un número primo, entonces tiene un divisor no trivial $\leq\sqrt{N}$.

(*a*) Modifique el programa, de manera que solamente se examine si N es divisible por cualquiera de los enteros 2, 3,..., $\sqrt{N}$.
(*b*) Modifique el programa, de manera que solamente se examine si N es divisible por 2 ó por cualquier entero impar $\leq\sqrt{N}$.

4.49 Traduzca a Fortran la siguiente proposición:

SI $1 \leq X \leq 2$ ENTONCES K = 1
DE LO CONTRARIO K = 2

(Sugerencia: La condición $1 \leq X \leq 2$ significa que $1 \leq X$ y $X \leq 2$ y se puede implementar con varias proposiciones IF lógica.)

Respuestas a los problemas suplementarios seleccionados

4.16
(*a*) IF(A.GT.B) STOP
(*b*) IF(J.EQ.K + 3) GO TO 20
(*c*) IF((A + B**2).LT.100.0) GO TO 30
(*d*) IF(X – Y.GE.50.0) GO TO 40
(*e*) IF(I.NE.4) GO TO 50
(*f*) IF(J.LE.K) STOP

4.17
(*a*) .EQ. en lugar de = .
(*b*) .GT. en lugar de GT.
(*c*) B^2–AC no es una expresión de relación.
(*d*) X.LT.Y + Z no es una expresión aritmética.
(*e*) 100.0 en lugar de 100 si X es real.
(*f*) No hay error, aunque es preferible escribir A–100.0.
(*g*) No hay error.
(*h*) Falta un paréntesis entre AMOUNT y STOP; INTEREST tiene más de seis caracteres; INTEREST y AMOUNT tienen tipos diferentes.

4.18 (*a*)
```
      IF(A + B**2.GT.100.0) TO TO 100
200   .....................................
```
ó
```
      IF(A + B**2 - 100.0) 200, 200, 100
200   .....................................
```

(*b*)
```
      IF(A - B.LE.100.0) GO TO 10
      K = 2
      GO TO 20
10    K = 1
20    .....................................
```
ó
```
      IF(A - B - 100.0) 10, 10, 50
50    K = 2
      GO TO 20
10    K = 1
20    .....................................
```

(*c*)
```
IF(J.LT.K) GO TO 10
IF(J.EQ.K) GO TO 20
IF(J.GT.K) GO TO 30
```
ó
```
IF(J - K) 10, 20, 30
```

4.19 (*a*)
```
      IF(X - Y) 20, 20, 10
10    S
20    T
```
(*b*)
```
      IF(I - J) 20, 10, 20
10    S
20    T
```
(*c*)
```
      IF(X - Y) 10, 10, 20
10    S
20    T
```
(*d*)
```
      IF(A - B) 20, 10, 10
10    S
20    T
```

4.20 (*a*) 5 (*b*) 7 (*c*) 7 (*d*) 8 (*e*) 10 (*f*) 5

4.21 (*a*) 9.0 y 3.0 (*b*) 5.0 y 3.0

4.22 (*a*) 5.0 (*b*) 10.0 (*c*) 8.0 (*d*) 5.0

4.23 (*a*) No hay coma antes de JIM.
(*b*) No se permite el 0.
(*c*) No hay errores.
(*d*) 2345678 es muy grande como número de proposición.
(*e*) TOM debe ser una variable entera.

4.24
```
IF(K.EQ.1) GO TO 47
IF(K.EQ.2) GO TO 33
IF(K.EQ.3) GO TO 55
GO TO 77
```

4.25
```
IF(JIM - 2) 20, 30, 40
```

4.26 (*a*) 41 (*b*) 31 (*c*) 51 (*d*) El error consiste en que TYPE contiene el valor 5.

Capítulo 5

Ciclos DO

5.1 INTRODUCCION

La transferencia de controles que estudiamos en el capítulo 4 nos permitía ejecutar un grupo de instrucciones repetidamente. Recordemos que se utilizó un contador I junto con una proposición IF para salir de los ciclos discutidos en las secciones 4.5 y 4.6. Este mecanismo de control para repetir un proceso N veces se ilustra en el diagrama de flujo de la figura 5-1. Esto es, el contador se coloca inicialmente en 1. Después de cada iteración este contador se incrementa en 1 y luego se hace la pregunta "¿I ≤ N?" antes de que el ciclo se repita nuevamente. El ciclo termina una vez que I excede N

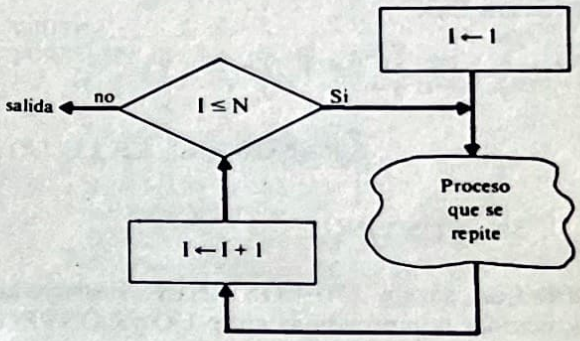


Fig. 5-1

Puesto que la repetición de un proceso es fundamental en la escritura de programas de computador, sería sumamente útil tener una orden semejante a una macro tal como: HAGA ... MIENTRAS 1 ≤ I ≤ N,

HAGA

[Proceso que se repite] RANGO DEL DO

MIENTRAS 1 ≤ I ≤ N

o una orden semejante a una macro tal como: HAGA PARA 1 ≤ I ≤ N... REPITA,

HAGA PARA 1 ≤ I ≤ N

[Proceso que se repite] RANGO DEL DO

REPITA

El Fortran no estructurado contiene tal tipo de orden —la proposición DO—. Como la proposición DO es una de las más poderosas construcciones del Fortran, este capítulo se dedica completamente a la discusión de esta instrucción semejante a una macro. (Por una instrucción semejante a una macro, se entiende una instrucción simple que involucra una lista predeterminada de instrucciones.)

El Fortran estructurado, que se discute en el capítulo 12, contiene otros tipos de orden DO.

5.2 PROPOSICION CONTINUE

Presentamos primero una nueva proposición ejecutable, la proposición CONTINUE, la cual simplemente consta de la palabra

CONTINUE

Las razones para utilizar esta proposición se aclararán más adelante en la sección 5.8, pero su función está descrita exactamente por su nombre —continuar la ejecución—. La proposición CONTINUE se denomina también proposición ejecutable *ficticia*, puesto que no genera instrucciones en lenguaje de máquina.

La mayoría de los compiladores permiten el uso libre de la proposición CONTINUE en cualquier parte de un programa; sin embargo, algunos requieren que las proposiciones CONTINUE se rotulen.

5.3 USOS SENCILLOS DE LA PROPOSICION DO

Para repetir un proceso N veces, se puede utilizar frecuentemente la siguiente forma acostumbrada de la proposición DO:

```
DO 50 I = 1, N
        [        :
        [ RANGO DEL CICLO DO ]
        [        :
50  CONTINUE
```

La anterior pareja DO-CONTINUE instruye al computador para que ejecute repetidamente las proposiciones comprendidas entre DO y CONTINUE (denominadas el *rango* del ciclo DO). A la variable entera I (llamada la *variable índice*) se le asigna inicialmente el valor 1, y después de cada iteración, el valor de I se incrementa en 1. La iteración se continúa (se repite) mientras $I \leq N$. Cuando $I > N$, se ejecuta enseguida la proposición que sigue a la proposición CONTINUE. (Nótese que el rótulo de la proposición CONTINUE se escoge arbitrariamente.)

Nota. Para que la lectura sea más fácil, generalmente el rango del DO se escribe más hacia adentro que los otros renglones. Esto no afecta al programa porque el computador ignora los espacios en blanco en las proposiciones Fortran.

Recalquemos que el significado exacto de la anterior proposición DO se ilustra en el diagrama de flujo de la figura 5-2(*a*). Puesto que una proposición DO comprende muchas cajas, por conveniencia, introducimos una macro-caja equivalente, como se muestra en la figura 5-2(*b*).

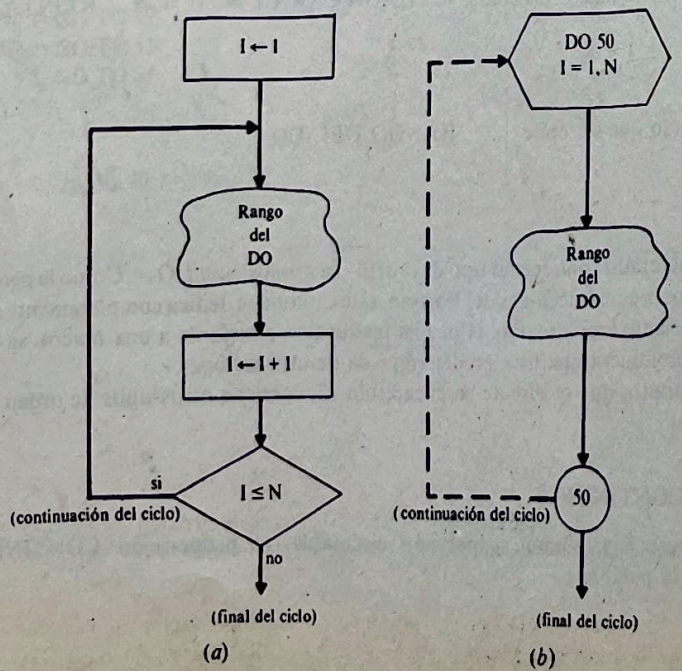
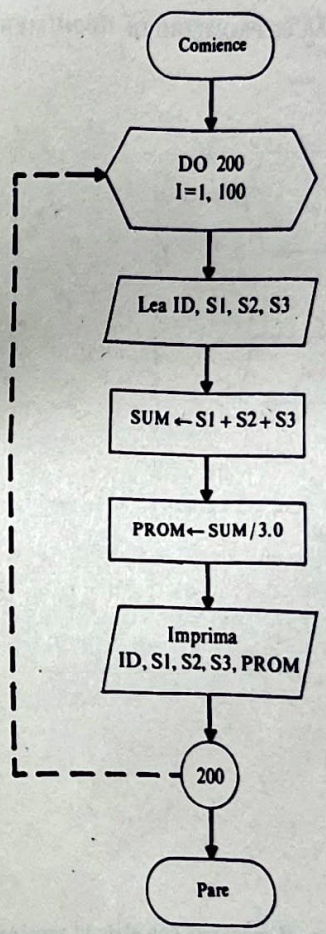


Fig. 5-2

EJEMPLO 5.1

Recordemos el ejemplo 4.2(*b*) que calcula el promedio de los resultados de tres exámenes para cada estudiante de un curso de 100 estudiantes. (Los tres resultados del examen y el número ID de identificación de cada estudiante se perforan en una tarjeta de datos.)

Reescribimos el programa utilizando una proposición DO. El diagrama de flujo del programa y su traducción Fortran aparecen en la figura 5-3. Obsérvese cómo resalta claramente el cuerpo del ciclo DO, ya que se ha escrito más adentro que los otros renglones.



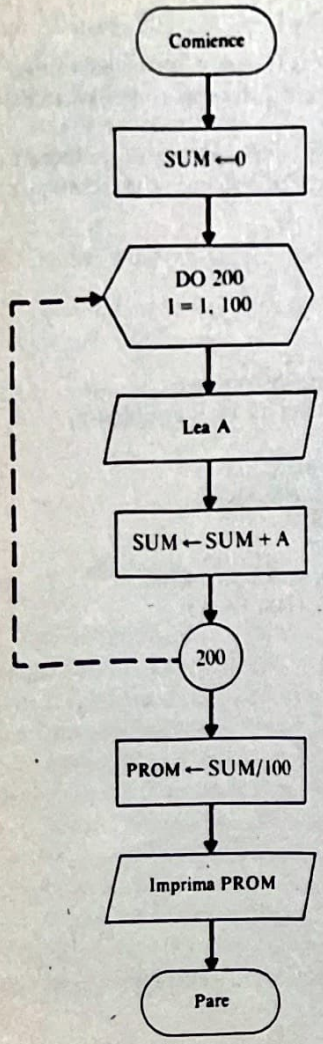

```
      WRITE(6, 5)
    5 FORMAT('1', 3X, 'ID NO.', 4X, 'TEST 1',
     1     4X, 'TEST 2', 4X, 'TEST 3', 3X, 'PROMEDIO')
       DO 200 I = 1,100
            READ(5, 10) ID, S1, S2, S3
   10       FORMAT(5X, I5, 3(4X, F6.2) )
            SUM = A1 + S2 + S3
            PROM = SUM/3.0
            WRITE(6, 20) ID, S1, S2, S3, PROM
   20       FORMAT(5X, I5, 4 (4X, F6.2) )
  200 CONTINUE
      STOP
      END
```

Fig. 5-3

EJEMPLO 5.2

En la sección 4.9 del capítulo 4, se escribió un programa para encontrar el promedio de 100 números. Reescribiremos este programa utilizando la versión simple de un ciclo DO. El diagrama de flujo y la traducción Fortran aparecen en la figura 5-4.

Obsérvese que el índice I del ciclo DO en el ejemplo anterior actúa simplemente como un contador, el cual acumula el número de veces que pasamos a través del ciclo. Realmente, el ciclo DO es mucho más poderoso, ya que el índice I está disponible para cálculos dentro de él. Esto se ilustra en el próximo ejemplo.




```
C
C     PROGRAMA PARA ENCONTRAR EL PROMEDIO DE 100 NUMEROS
C
      SUM = 0.0
      DO 200 I = 1, 100
            READ(5, 20) A
   20       FORMAT(F10.2)
            SUM = SUM + A
  200 CONTINUE
      PROM = SUM/100.0
      WRITE(6, 10) PROM
   10 FORMAT(1X, F10.2)
      STOP
      END
```

Fig. 5-4

EJEMPLO 5.3

(*a*) Escriba un programa que imprima los 100 primeros enteros positivos. (Compárese con el de la sección 4.3.) El diagrama de flujo aparece en la figura 5-5(*a*); su traducción Fortran es la siguiente:

```
      DO 500 I = 1, 100
            WRITE(6, 10) I
   10       FORMAT(10X, I3)
  500 CONTINUE
      STOP
      END
```

(*b*) Escriba un programa que encuentre la suma de los 30 primeros enteros positivos. Deseamos calcular $1 + 2 + 3 + \ldots + 30$. Consideremos primero ISUM = 0 y luego ejecutamos repetidamente

```
ISUM = ISUM + I
```

para I = 1, 2,..., 30 utilizando un ciclo DO. El diagrama de flujo aparece en la figura 5-5(*b*); su traducción Fortran es como sigue:

```
      ISUM = 0
      DO 700 I = 1, 30
            ISUM = ISUM + I
700   CONTINUE
      WRITE(6, 20) ISUM
 20   FORMAT(10X, I10)
      STOP
      END
```

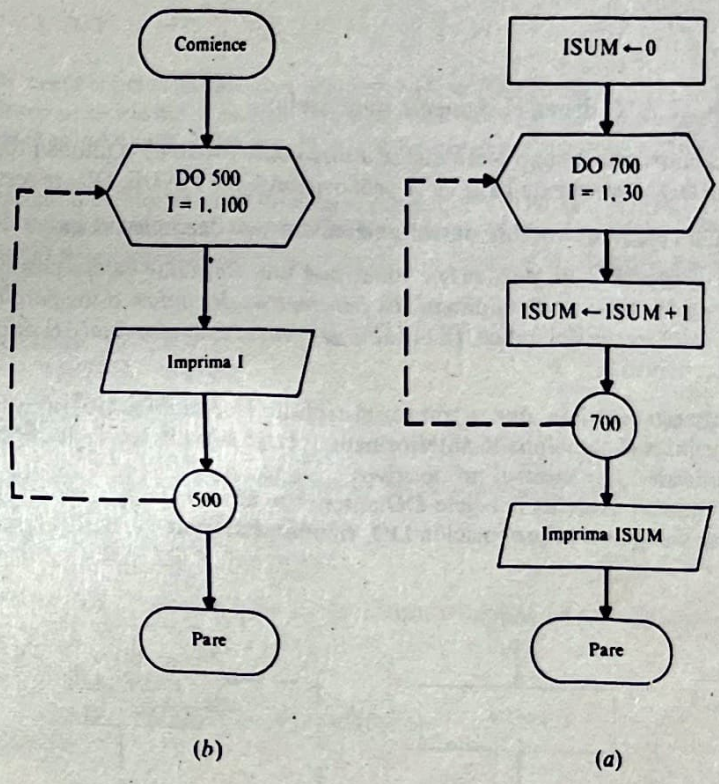


Fig. 5-5

(*c*) Escriba un segmento de programa que calcule el producto de los 10 primeros enteros positivos. El esquema es muy parecido al de (*b*), excepto que deseamos calcular

$$1 \cdot 2 \cdot 3 \cdot \ldots \cdot 10$$

Esto lo hacemos considerando primero IPROD = 1 y ejecutando repetidamente

```
IPROD = IPROD*I
```

por medio de un ciclo DO con índice I. El segmento de programa es el siguiente:

```
      IPROD = 1
      DO 100 I = 1, 10
            IPROD = IPROD*I
100   CONTINUE
```

5.4 PROPOSICION DO

El ciclo DO tratado en la sección precedente es solamente un caso especial. El ciclo DO más general (no estructurado) tiene la siguiente forma:

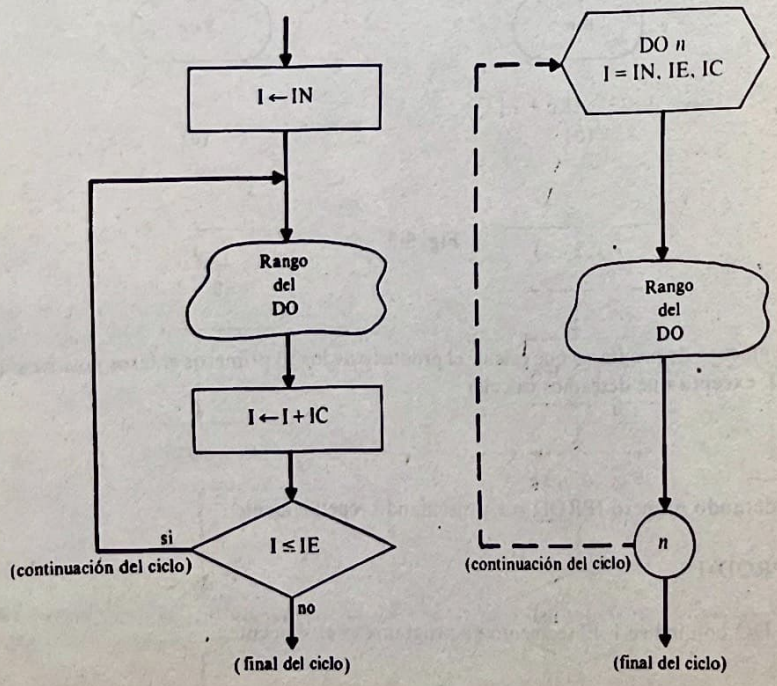
```
DO n I = IN, IE, IC
[ Rango del ciclo DO ]
n  CONTINUE
```

Los símbolos *n*, I, IN, IE e IC tienen el siguiente significado:

1. La *n* es un número de proposición que se utiliza como rótulo de la última proposición ejecutable en el ciclo DO, que en este caso es la proposición CONTINUE. (Véase sección 5.8.)
2. La I denota cualquier nombre de variable entera y se denomina el *índice* del ciclo.
3. Tanto IN como IE e IC pueden ser solamente una constante entera positiva o un nombre de variable entera. Ellas se denominan los *parámetros del índice* o los *parámetros del ciclo*. IN denota el *valor inicial* del índice, IE el *valor de control* o el *valor final* o el *valor límite* del índice, e IC es el *incremento*.

Nota. El Fortran estructurado, que se trata en el capítulo 12, contiene otros tipos de ciclos DO. Uno de tales ciclos DO es semejante al mencionado anteriormente, pero con la diferencia de que permite incrementos negativos y valores iniciales y de control no positivos.

El diagrama de flujo equivalente del ciclo DO anterior se muestra en la figura 5-6(*a*). (Véase también figura 5-2.) Cuando se encuentra la proposición DO, el índice I se toma igual al valor inicial IN. Luego, el



(*a*) (*b*)

Fig. 5-6

control pasa a través del rango del ciclo DO, es decir, desde la primera proposición que sigue a la proposición DO hasta la proposición CONTINUE. Ahora el índice I se aumenta en el valor del incremento IC y este nuevo valor de I es examinado. Si I excede al valor de control IE, entonces el ciclo DO se da por realizado y el control pasa a la primera proposición que sigue al ciclo DO (es decir, a la primera proposición después de la proposición CONTINUE). Si no lo excede el rango del ciclo DO se ejecuta de nuevo y así sucesivamente. El índice I toma un nuevo valor cada vez que el computador prosigue a través del ciclo DO. La figura 5-6(*b*) da el diagrama de flujo equivalente, utilizando la macro-caja para el ciclo DO.

EJEMPLO 5.4

(*a*) Supongamos que el computador encuentra la proposición DO:

```
DO 200 K = 2, 10, 3
```

El computador puede repetir el ciclo DO tres veces, primero con K = 2, luego con K = 2 + 3 = 5 y por último con K = 5 + 3 = 8. Después de repetir el ciclo tres veces, K se incrementa a 8 + 3 = 11, que excede al valor límite 10; por lo tanto, el computador no repite el ciclo DO una cuarta vez. Esto muestra que el índice no necesita ser igual al valor de control (límite).

(*b*) Supongamos que queremos imprimir los enteros pares entre 2 y 100, esto es, 2, 4, 6, . . . , 98, 100. Inicialmente, consideramos IPAR = 2 y ejecutamos repetidamente IPAR = IPAR + 2, mientras IPAR ≤ 100. El diagrama de flujo aparece en la figura 5-7(*a*). Si utilizamos un ciclo DO, se produce el diagrama de flujo de la figura 5-7(*b*) y su traducción Fortran es como sigue:

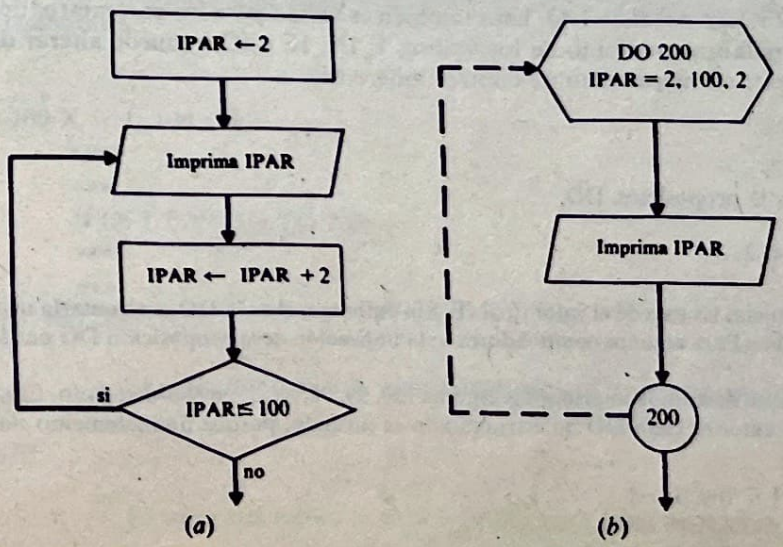
```
      DO 200 IPAR   = 2, 100, 2
             WRITE(6, 10) IPAR
10           FORMAT(6X, I3)
200   CONTINUE
```

(*c*) Supongamos que A está en la memoria. Calcular los productos:

(*i*) (1 + A)(1 + 2A)(1 + 3A) . . . (1 + 10A)
(*ii*) (1 + A)(1 + 4A)(1 + 7A) . . . (1 + 19A)

Como en el ejemplo 5.3(*c*), el producto (*i*) se puede calcular utilizando un ciclo DO como sigue:

```
      PROD = 1.0
      DO 20 I = 1, 10
             PROD = PROD*(1.0 + FLOAT(I)*A)
20    CONTINUE
```



(*a*) (*b*)

Fig. 5-7

El producto (*ii*), con una modificación menor, se puede calcular de una manera análoga, como sigue:

```
      PROD = 1.0
      DO 50 I = 1, 19, 3
            PROD = PROD*(1.0 + FLOAT(I)*A)
50    CONTINUE
```

5.5 REGLAS ACERCA DEL USO DEL CICLO DO

Las siguientes reglas se deben tener en cuenta en la aplicación del ciclo DO no estructurado, que vimos anteriormente.

1. Si el incremento es 1, éste puede omitirse. Por ejemplo,

   ```
   DO 100 I = 1, N, 1
   ```

   puede escribirse en la forma abreviada

   ```
   DO 100 I = 1, N
   ```

2. En cualquier proposición DO, cada uno de los parámetros del índice: IN, IE, IC debe ser una constante entera *positiva* o una variable entera, simple, no subindicada, que sólo adopte valores positivos. De este modo,

   ```
   DO 200 I = 20, 1, -2
   ```

   es inválida. (Véase nota de la página 118.)

Los dos puntos siguientes merecen especial atención, puesto que ellos conducen a errores frecuentes.

3. En la proposición DO en sí, no se pueden ejecutar cálculos aritméticos. Por lo tanto,

   ```
   DO 300 I = M, K + 3, 2
   ```

   es inválida. Sin embargo, se puede llevar a cabo el mismo cálculo aritmético si se introduce una nueva variable, digamos KK, y entonces se escribe

   ```
   KK = K + 3
   DO 300 I = M, KK, 2
   ```

4. Aunque el índice I está disponible para cálculos dentro del ciclo DO, él no puede modificarse dentro del rango del ciclo DO. Esto también es válido para los parámetros del índice IN, IE e IC. En otras palabras, ninguno de los valores I, IN, IE e IC se puede alterar dentro del ciclo DO (excepto por su mecanismo de control inherente).

EJEMPLO 5.5

(*a*) Consideremos la proposición DO

```
DO 200 I =  4, 2, 3
```

Obsérvese que el valor inicial IN excede el valor final IE. Sin embargo, el ciclo DO se ejecutaría una vez, de acuerdo con nuestro diagrama de flujo. (Para algunos compiladores, esta utilización de la proposición DO puede generar un mensaje de error.)

(*b*) Supongamos que deseamos imprimir los enteros 100, 99, 98,..., 1, en el orden dado. El siguiente segmento de programa que utiliza el anterior ciclo DO no estructurado es inválido, porque un incremento no puede ser negativo:

```
      DO 500 I = 100, 1, -1
            WRITE(6, 20) I
20          FORMAT(1X, I3)
500   CONTINUE
```

Sin embargo, es posible escribir dicho programa utilizando un ciclo DO no estructurado. Notemos primero que podemos usar la proposición

```
DO 500 I = 1, 100
```

Sin embargo, cuando I es igual a 1, queremos imprimir 100; cuando I = 2, deseamos imprimir 99 y así sucesivamente. Esto es:

Valor de I:	1, 2, 3, ..., 99, 100
Valor a imprimir:	100, 99, 98, ..., 2, 1

Si consideramos que J representa el valor que queremos imprimir, observemos que I + J siempre es igual a 101. Por lo tanto, J = 101 − I. De esta manera, el siguiente segmento de programa, que utiliza un ciclo DO no estructurado, lleva a cabo nuestra tarea:

```
      DO 500 I = 1, 100
            J = 101 - I
            WRITE(6, 20) J
20          FORMAT(1X, I3)
500   CONTINUE
```

5.6 SALIDA DE UN CICLO DO

Hay dos maneras de salir de cualquier ciclo DO: (*a*) salida normal, (*b*) salida anormal. Veámoslas:

(*a*) Salida normal

Una salida normal tiene lugar cuando el índice I excede al valor de control IE [tal como se indicó en el diagrama de flujo de la figura 5-6(*a*)]. En ese caso, el control se transfiere a la primera proposición ejecutable que sigue al ciclo DO. Una propiedad adicional de la salida normal de un ciclo DO es que el valor del índice I en el momento de la salida *no está definido* y por lo tanto, no se puede utilizar en ningún cálculo posterior. Todos los ejemplos que se han presentado hasta ahora, tienen que ver con salidas normales.

(b) Salida anormal

Es posible hacer una transferencia desde el interior hacia el exterior de un ciclo DO, por ejemplo, por medio de una proposición IF en el ciclo DO, aun cuando el valor actual del índice I no exceda al valor de control IE. Esta salida se denomina una *salida anormal*. Una propiedad fundamental de una salida anormal de un ciclo DO es que, en el momento de la salida, el valor actual del índice I se conserva y, en consecuencia, puede ser utilizado en cálculos adicionales o en operaciones de E/S.

El siguiente esquema de programa, que utiliza a K como índice del DO ilustra cuándo está definido el valor del índice K y cuándo no.

```
      DO 100 K = 1, 100, 2
            ***
            ***
            IF(X.LT.Y) GO TO 200
            ***
            ***
100   CONTINUE
      ***
      ***     } El valor del índice K no está definido para estas proposiciones.
      ***
      STOP
200   ***
      ***     } El valor del índice K está definido para estas proposiciones.
      ***
      STOP
```

EJEMPLO 5.6

Escriba un programa que lea un entero K > 2 y determine si K es un número primo. Si no lo es, imprima un divisor no trivial de K. (Compárese con el ejemplo 4.5.)

Hemos establecido que si K no es un número primo, entonces K tiene un divisor no trivial $\leq K/2$. El diagrama de flujo y el programa aparecen en la figura 5-8. Notemos que el índice del ciclo varía desde 2 hasta K/2. Puesto que ninguna operación aritmética puede tener lugar en una proposición DO, la proposición

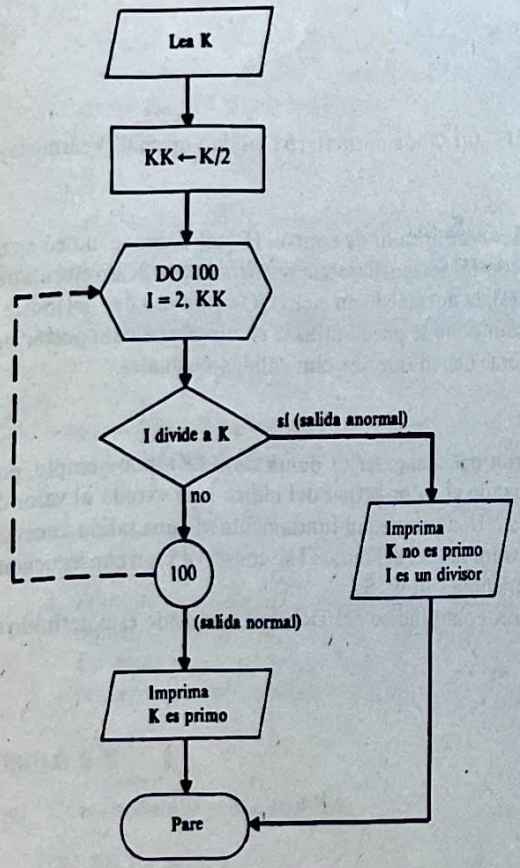
```
KK = K/2
```

entonces deberá utilizarse al comienzo.

Obsérvese que el ciclo DO examina cada uno de los enteros 2, 3,..., KK para comprobar si ellos dividen a K. Si no lo hacen, entonces el ciclo DO se continúa. De esta manera, se tiene una salida normal cuando ninguno de los números 2, 3,..., KK son divisores de K; en este caso, K es un número primo. Por otra parte, una vez que I divida a K, tendremos una salida anormal. En esta ocasión, el valor del índice I está definido y, por lo tanto,

```
WRITE(6, 20) K, I
```

puede ejecutarse.




```
      READ(5, 8) K
    8 FORMAT(I10)
      KK = K/2
      DO 100 I = 2, KK
            IF(K.EQ.(K/I)*I) GO TO 50
  100 CONTINUE
      WRITE(6, 10) K
   10 FORMAT(10X, I5, 1X, 'ES PRIMO')
      STOP
   50 WRITE(6, 20) K, I
   20 FORMAT(10X, I5, 1X, 'NO ES PRIMO'/
     1          10X, I5, 1X, 'ES UN DIVISOR')
      STOP
      END
```

Fig. 5-8

5.7 TRANSFERENCIA DENTRO DE Y HACIA UN CICLO DO

Siempre se puede hacer una transferencia desde un punto en un ciclo DO hacia otro punto en el *mismo* ciclo DO; asimismo, siempre se puede salir libremente de cualquier ciclo DO. Sin embargo, *no es posible* entrar en una proposición del rango del ciclo DO. Es decir, que la única manera de hacer una transferencia hacia las proposiciones incluidas dentro de un ciclo DO, es por la vía normal de la proposición DO. Es importante recordar, sin embargo, que cada vez que se transfiere el control a una proposición DO, el índice del ciclo DO vuelve a su valor inicial.

Supongamos que queremos un segmento de programa Fortran que utilice un ciclo DO para calcular la suma

$$1 + 2 + 3 + 5 + 6 + 7 + 8 + 9 + 10$$

esto es, la suma de los 10 primeros enteros positivos excluyendo al entero 4. Consideremos los dos segmentos de programa que siguen:

```
(A)       ISUM = 0
          DO 100 I = 1, 10
                IF(I.EQ.4) I = I + 1
                ISUM = ISUM + I
      100 CONTINUE

(B)       ISUM = 0
       50 DO 200 I = 1, 10
                IF(I.EQ.4) GO TO 50
                ISUM = ISUM + I
      200 CONTINUE
```

El programa A es inválido porque se modifica el índice I en el ciclo DO. Esto es, si las proposiciones

```
IF(I.EQ.4) I = I + 1
ISUM = ISUM + I
```

no estuvieran en un ciclo DO, entonces el valor 5 se sumaría a ISUM, si I originalmente tuviera el valor 4. Sin embargo, estas proposiciones no se permiten, puesto que aquí estamos en el intervalo de un ciclo DO.

Por el contrario, el programa B utiliza correctamente el ciclo DO. Sin embargo, la lógica del programa no es correcta. Es decir, hay una salida anormal cuando I toma el valor 4, pero el control se retorna a la proposición DO. Tal como se mencionó anteriormente, cada vez que se ejecuta la proposición DO, el índice vuelve a su valor inicial, que en este caso es 1. Por lo tanto, este programa dará un ciclo infinito que calculará

$$1 + 2 + 3 + 1 + 2 + 3 + 1 + 2 + 3 + \cdots$$

y así sucesivamente.

El propósito del programa B es saltar ISUM = ISUM + I cuando I tenga el valor 4. No obstante, queremos que el ciclo DO continúe su desarrollo normal. Esto puede llevarse a cabo, transfiriendo el control a la proposición CONTINUE y no a la proposición DO. O sea, que el programa B calculará la suma requerida, si

```
IF(I.EQ.4) GO TO 50
```

se reemplazará por

```
IF(I.EQ.4) GO TO 200
```

Este ejemplo ilustra el punto principal de la próxima sección; la necesidad de la proposición CONTINUE.

5.8 NECESIDAD DE LA PROPOSICION CONTINUE

Realmente *no se requiere* que la última proposición de cualquier ciclo DO sea la proposición CONTINUE, pero sí *debe ser una proposición ejecutable*, diferente de una proposición GO TO, de una proposición IF aritmética o de otra proposición DO. Por ejemplo, en lugar de

```
      ISUM = 0
      DO 200 I = 1, 100
            ISUM = ISUM + I
200   CONTINUE
```

se puede escribir

```
      ISUM = 0
      DO 200 I = 1, 100
200         ISUM = ISUM + I
```

Sin embargo, impulsaremos el uso de la pareja DO-CONTINUE para prevenir el siguiente tipo de error común:

```
      DO 300 I = 1, 100
            WRITE(6, 300) I
300         FORMAT(10X, I3)
```

(El error consiste en el uso de la proposición FORMAT, que no es ejecutable, como última proposición en el ciclo DO. La proposición WRITE, en lugar de la proposición FORMAT, deberá rotularse como 300.) Más importante todavía, es que la pareja DO-CONTINUE ofrece la ventaja pedagógica de normalizar la forma de cualquier ciclo DO; la proposición CONTINUE, a su vez, sirve como delimitadora del ciclo DO.

Por otra parte, hay una situación en que se debe utilizar la proposición CONTINUE. Supongamos que dentro de un ciclo DO hay una proposición condicional, en donde las dos alternativas no tienen nada en

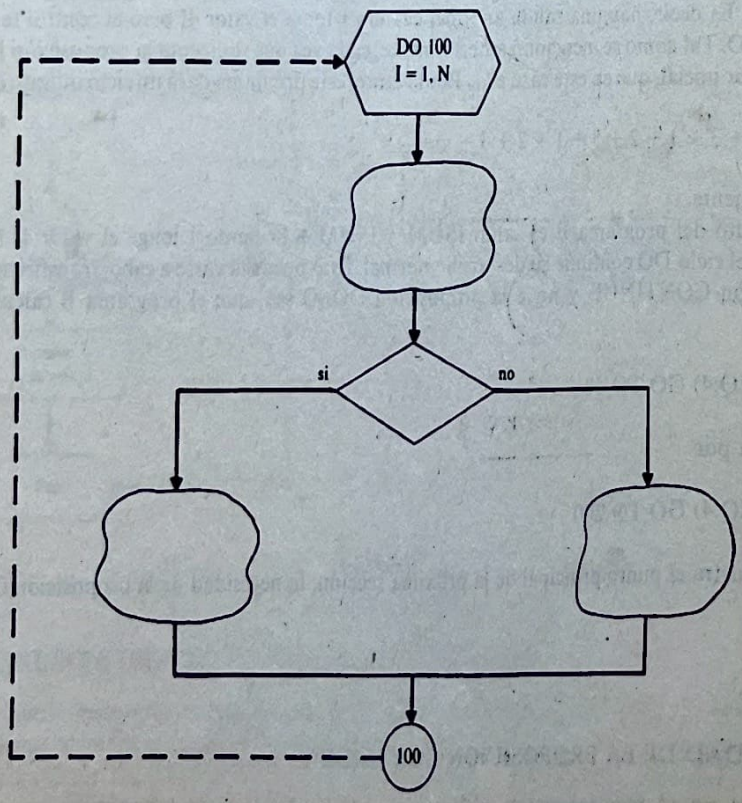


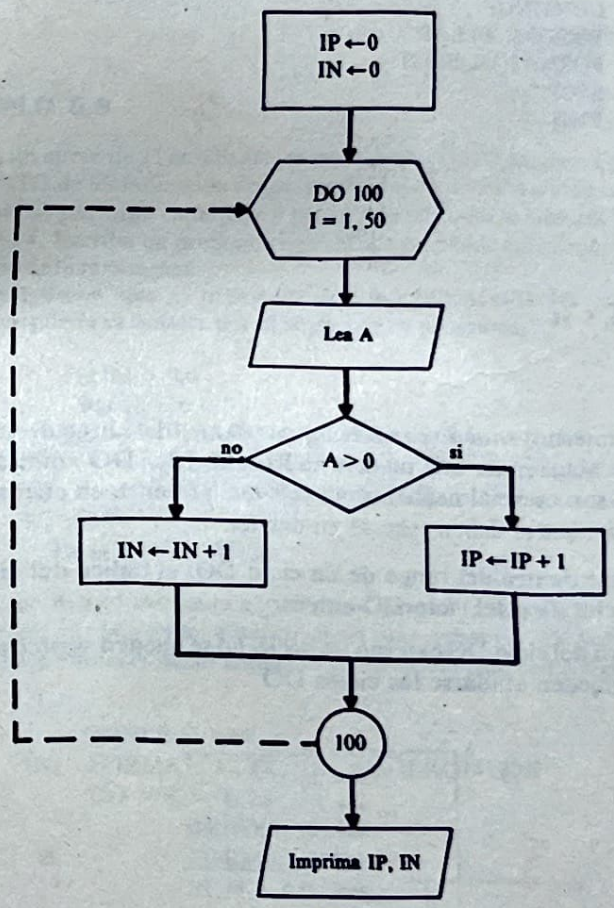
Fig. 5-9

común, tal como se ilustra en la figura 5-9. Se requiere una clase especial de proposición ejecutable, de tal manera que ambas alternativas permitan que el ciclo DO prosiga su curso normal. Para este propóstio se creó la proposición CONTINUE. Un ejemplo sobre el particular es el segmento de programa B corregido en la sección 5.7.

EJEMPLO 5.7

Se suministra un paquete de 50 tarjetas en cada una de las cuales se perfora un número diferente de cero. Escriba un programa Fortran que cuente la cantidad de números positivos y de números negativos.

El diagrama de flujo para el programa y su traducción Fortran aparecen en la figura 5-10. Aquí, tendremos que utilizar de nuevo una proposición CONTINUE.



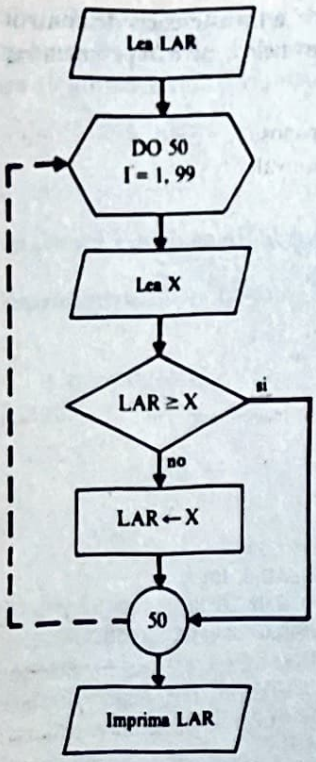

```
      IP = 0
      IN = 0
      DO 100 I = 1, 50
             READ(5, 10) A
10           FORMAT(F10.2)
             IF(A.GT.0.0) GO TO 50
             IN = IN + 1
             GO TO 100
50           IP = IP + 1
100   CONTINUE
      WRITE(6, 20) IP, IN
20    FORMAT(6X, 'POSITIVO = ', I3/
     1       6X, 'NEGATIVO =', I3)
      STOP
      END
```

Fig. 5-10

EJEMPLO 5.8

Escriba un programa para encontrar el mayor de 100 números, si cada uno de ellos se perfora en una tarjeta separada.

El diagrama de flujo del programa y su traducción Fortran aparecen en la figura 5-11. Obsérvese que el valor final del índice en la proposición DO es 99 y no 100. Esto proviene del hecho de que el primer número se lee como LAR antes de que se ejecute el ciclo DO. Nuevamente, tenemos que utilizar una proposición CONTINUE.




```
C
C       PROGRAMA PARA ENCONTRAR EL MAYOR DE 100 NUMEROS
C
        REAL LAR
        READ(5, 10) LAR
10      FORMAT(F10.2)
        DO 50 I = 1, 99
              READ(5, 10) X
              IF(LAR.GE.X) GO TO 50
              LAR = X
50      CONTINUE
        WRITE(6, 20) LAR
20      FORMAT(1X, F10.2)
        STOP
        END
```

Fig. 5-11

5.9 CICLOS DO ANIDADOS

Es posible tener un ciclo DO (el ciclo DO interno) *situado completamente* dentro del rango de otro ciclo DO (el ciclo DO externo). Los ciclos DO que ocurren de esta manera se llaman ciclos DO anidados. Las reglas que se aplican a los ciclos DO anidados son esencialmente las mismas que se tienen en cuenta en los ciclos DO simples. Sin embargo, los puntos siguientes deben tenerse en cuenta:

1. Como el índice no puede redefinirse dentro del rango de un ciclo DO, el índice del ciclo DO interno no debe ser el mismo que el índice del ciclo DO externo.
2. El ciclo DO interno debe estar dentro del ciclo DO externo, es decir, no se pueden sobreponer. La figura 5-12 ilustra la forma como pueden anidarse los ciclos DO

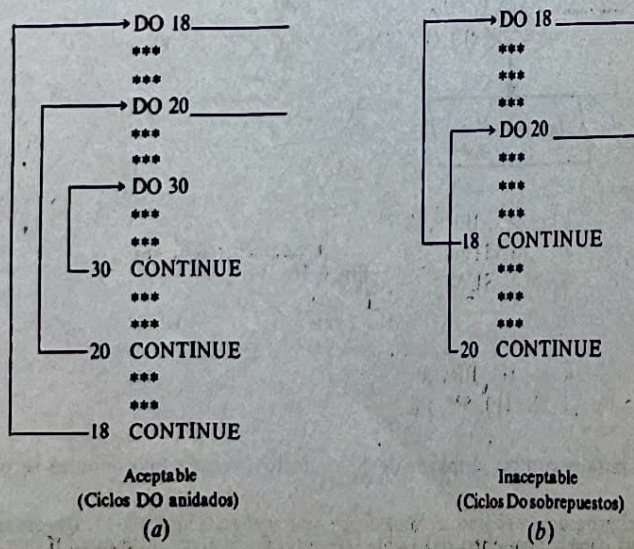


Fig. 5-12

3. El número de ciclos DO anidados está limitado por el compilador. La transferencia de control dentro de los ciclos DO anidados, se puede hacer por medio de artificios; pero, en esencia las reglas para los ciclos DO simples se mantienen.

Por último, notamos que los ciclos DO anidados pueden tener la misma proposición final. Así, la figura 5-13 muestra los ciclos DO anidados que son aceptables y que pueden ser equivalentes.

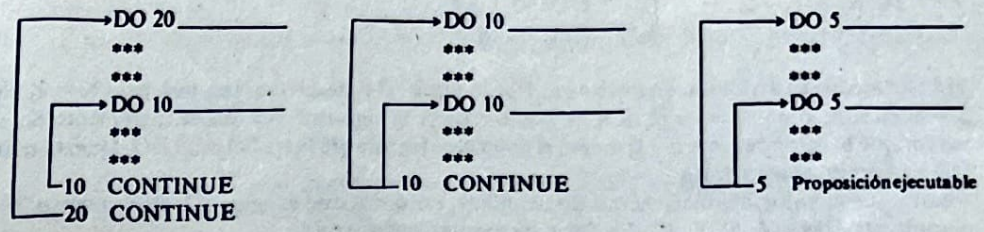


Fig. 5-13

EJEMPLO 5.9

En un curso de 25 estudiantes se practican tres (3) exámenes. Los números ID de identificación de los estudiantes y los resultados de los exámenes se perforan en tarjetas y se arreglan tal como lo muestra la figura 5-14. Escriba un programa que calcule para cada estudiante el promedio de los exámenes.

Observemos que el promedio de los resultados de los tres exámenes puede calcularse por el segmento de programa:

```
      SUM = 0.0
      DO 20 I = 1, 3
            READ(5, 10) RESUL
10          FORMAT(F6.2)
            SUM = SUM + RESUL
20    CONTINUE
      PROM = SUM/3.0
```

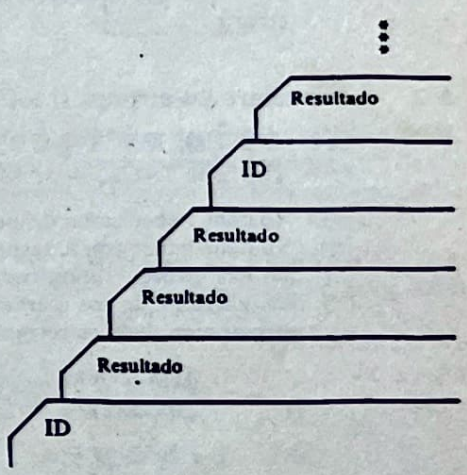


Fig. 5-14

Puesto que hemos calculado el promedio para cada uno de los 25 estudiantes, el segmento anterior tiene que repetirse 25 veces. Encerramos entonces dicho segmento en otro ciclo DO, como sigue:

```
      WRITE(6, 100)
100   FORMAT('1', 4X, 'ID', 9X, 'PROMEDIO')
      DO 80 K = 1, 25
            READ(5, 90) ID
90          FORMAT(I10)
            SUM = 0.0
            DO 20 I = 1, 3
                  READ(5, 10) RESUL
10                FORMAT(F6.2)
                  SUM = SUM + RESUL
20          CONTINUE
            PROM = SUM/3.0
            WRITE(6, 30) ID, PROM
30          FORMAT(1X, I10, 5X, F6.2)
80    CONTINUE
```

Segmento anterior

Nota: Obsérvese el desplazamiento del ciclo DO interno dentro del ciclo DO externo. Nuevamente, esto se ha hecho para facilitar la lectura, lo que no afecta la ejecución del programa.

Problemas resueltos

CICLOS DO

5.1 Determine el número de veces y los valores del índice para los cuales se ejecutan cada uno de los siguientes ciclos DO.

(*a*) DO 10 K = 1, 11, 3 (*c*) DO 30 LARGE = 8, 18, 15
(*b*) DO 20 JIM = 4, 12 (*d*) DO 40 M = 7, 4, 2

(*a*) El valor inicial es 1 y el incremento es 3. Por lo tanto, el ciclo DO se ejecutará para K = 1, para K = 1 + 3 = 4, para K = 4 + 3 = 7 y para K = 7 + 3 = 10. A la siguiente vez que se incremente K, su valor será mayor que el valor de control 11; luego, el control se transferirá fuera del ciclo DO. De esta manera, el ciclo DO se ejecuta cuatro veces.

(*b*) Puesto que el valor del incremento no se indica, entonces este es igual a 1. Por lo tanto, el ciclo DO se ejecuta par JIM = 4, 5,..., 12. Es decir, se ejecuta nueve veces.

(*c*) El ciclo DO solamente se ejecuta una vez para LARGE = 8, ya que el siguiente valor de LARGE es 8 + 15 = 23 que es mayor que el valor de control 18.

(*d*) A pesar de que el valor inicial de M es mayor que el valor de control, el ciclo DO se ejecuta una vez para M = 7, ya que el control tiene lugar al final del ciclo DO. (Algunos compiladores darán un mensaje de error.)

5.2 Encuentre los errores, si los hay, en cada una de las proposiciones DO:

(*a*) DO 100, K = 3, 24, 2 (*c*) DO 300 JILL = JOHN, JIM
(*b*) DO 200 JACK = 5, M, 4, (*d*) DO 400 K = 4, 2*M, 3

(*a*) No debe haber coma después de 100.
(*b*) No debe haber coma después de 4.
(*c*) No hay errores (suponiendo que JOHN y JIM están definidas).
(*d*) En ninguno de los parámetros del ciclo DO pueden llevarse a cabo operaciones aritméticas. La proposición dada se reemplazaría, entonces, por las dos siguientes proposiciones:

```
MM = 2*M
DO 400 K = 4, MM, 3
```

5.3 Encuentre el valor final de K después de que se ejecute cada uno de los segmentos de programa Fortran:

(*a*)
```
      K = 2
      DO 10 I = 3, 8, 2
            K = K + I
10    CONTINUE
      K = 2*K
```

(*b*)
```
      K = 2
      DO 20 I = 3, 8, 2
            K = K + I
            IF(K.GT.6) GO TO 30
20    CONTINUE
30    K = 2*K
```

(*c*)
```
      K = 2
      M = 2
      DO 40 I = 3, 8, M
            K = K + I
40    CONTINUE
      K = 2*K
```

(*d*)
```
      K = 2
      DO 50 I = 3, 8, K
            K = K + I
50    CONTINUE
      K = 2*K
```

(*a*) La primera proposición asigna 2 a K. El ciclo DO se ejecuta entonces como sigue:

(*i*) Primero para I = 3. Esto da:

$$K \leftarrow K + I = 2 + 3 = 5$$

(*ii*) Luego para I = 5. Esto da:

$$K \leftarrow K + I = 5 + 5 = 10$$

(*iii*) Ahora para I = 7. Esto da:

$$K \leftarrow K + I = 10 + 7 = 17$$

El siguiente valor de I excede al valor de control; luego, el control se transfiere a la última proposición, la cual duplica el valor de K:

$$K \leftarrow 2*K = 2 \cdot 17 = 34$$

De esta manera, el valor final de K es 34.

(*b*) La primera proposición asigna 2 a K, y luego el ciclo DO se ejecuta como sigue:

(*i*) Primero para I = 3. Esto da:

$$K \leftarrow K + I = 2 + 3 = 5$$

Puesto que K < 6, el control no se transfiere a la proposición rotulada como 30.

(*ii*) Luego para I = 5. Esto da:

$$K \leftarrow K + I = 5 + 5 = 10$$

Puesto que K > 6, el control se transfiere a la proposición rotulada como 30, esto es, a la última proposición.

Esta proposición duplica el valor de K:

$$K \leftarrow 2*K = 2 \cdot 10 = 20$$

Por lo tanto, el valor final de K es 20.

(*c*) Puesto que M = 2, se trata del mismo programa presentado en (*a*); por lo tanto, el valor final de K es 34.

(*d*) Como K = 2, la proposición DO es la misma de (*a*). Sin embargo, K ahora es un parámetro y su valor se modifica en el programa, lo cual no está permitido. De esta manera, el segmento de programa no puede ejecutarse.

5.4 Encuentre el valor final de K después de que se ejecute cada uno de los segmentos de programa Fortran.

(*a*)
```
      K = 2
   10 DO 20 I = 3, 8, 2
            IF(I.EQ.5) GO TO 20
            K = K + I
   20 CONTINUE
      K = 2*K
```

(*b*)
```
      K = 2
   10 DO 20 I = 3, 8, 2
            IF(I.EQ.5) GO TO 10
            K = K + I
   20 CONTINUE
      K = 2*K
```

(*a*) La primera proposición asigna 2 a K. Por lo tanto, el ciclo DO se ejecuta como sigue:

(*i*) Primero para I = 3. Puesto que I ≠ 5, se ejecuta la proposición K = K + I la cual nos da:

$$K \leftarrow K + I = 2 + 3 = 5$$

(*ii*) Luego para I = 5. Como I = 5, el control se transfiere a la proposición CONTINUE, la cual permite regresar al ciclo DO.

(*iii*) Ahora para I = 7. Puesto que I ≠ 5, se ejecuta la proposición K = K + I la cual da:

$$K \leftarrow K + I = 5 + 7 = 12$$

El siguiente valor de I excede al valor de control; entonces, el control se transfiere a la proposición que sigue al ciclo DO, la cual dobla el valor de K. Por lo tanto, el valor final de K es 24.

(*b*) La primera proposición asigna 2 a K. En consecuencia, se ejecuta el ciclo DO como sigue:

(*i*) Primero para I = 3. Puesto que I ≠ 5, se ejecuta la proposición K = K + I, la cual nos da:

$$K = K + I = 2 + 3 = 5$$

(*ii*) Ahora, para I = 5. Como I = 5, el control se transfiere a la proposición DO.

Como quiera que el control se transfiere a la proposición DO, el ciclo DO inicia todo nuevamente y hace I = 3. De esta manera, (*i*) e (*ii*) se repite una y otra vez. Esto da un ciclo infinito que permite hallar un valor final para K. (Recalcamos que si el control se transfiere a la proposición DO, el índice vuelve a su valor inicial.)

5.5 Encuentre los errores, si los hay, en cada uno de los siguientes programas. La flecha curva representa una transferencia de control.

(*a*)
```
C       PRIMER PROGRAMA
        .....
        .....
        DO 10 I = 1, 25
        .....
        .....
   10   CONTINUE
        .....
        .....
        END
```

(*b*)
```
C       SEGUNDO PROGRAMA
        .....
        DO 10 I = 1, 25
        .....
        DO 20 J = 1, 15
        .....
        .....
   10   CONTINUE
        .....
   20   CONTINUE
        .....
        .....
        END
```

(*c*)
```
C       TERCER PROGRAMA
        .....
        DO 10 I = 1, 25
        .....
        .....
        DO 20 J = 1, 15
        .....
        .....
   20   CONTINUE
        .....
   10   CONTINUE
        .....
        END
```

(*d*)
```
C       CUARTO PROGRAMA
        .....
        DO 10 I = 1, 25
        .....
        .....
        DO 20 J = 1, 15
        .....
        .....
   20   CONTINUE
        .....
   10   CONTINUE
        .....
        .....
        END
```

(*a*) El control no se puede transferir a una proposición del rango del ciclo DO.
(*b*) Los ciclos DO se sobreponen.
(*c*) El control no se puede transferir al interior de un ciclo DO interno desde un ciclo DO externo.
(*d*) No hay errores. Podemos hacer una transferencia a un ciclo DO externo desde un ciclo DO interno.

PROGRAMAS

5.6 Escriba un programa que imprima los enteros positivos desde 1 hasta 300, de tal manera que en la salida aparezcan tres números en una línea, así:

```
  1     2     3
  4     5     6
 .....................
298   299   300
```

Queremos imprimir I, I + 1 e I + 2 en cada línea, para I = 1, 4, 7,... e I ≤ 300. Por lo tanto, utilizamos un ciclo DO con el índice I variando de 1 a 300 con incremento 3. El programa es el siguiente:

```
      DO 100 I = 1, 300, 3
             J = I + 1
             K = I + 2
             WRITE(6, 10) I, J, K
   10        FORMAT(1X, 3(I5, 3X))
  100  CONTINUE
       END
```

5.7 Escriba un programa Fortran que lea un entero positivo impar y calcule (con tres lugares decimales) la suma:

(*a*) $1 + 1/2 + 1/3 + \cdots + 1/N$ (*c*) $1 - 1/2 + 1/3 - \cdots + 1/N$
(*b*) $1 + 1/3 + 1/5 + \cdots + 1/N$

(*a*) Primero, consideramos SUM = 0.0. Luego, utilizamos un ciclo DO con índice K = 1, 2, ..., N para sumar 1/K a SUM. El programa es como sigue:

```
      READ(5, 10) N
   10 FORMAT(I5)
      SUM = 0.0
      DO 100 K = 1, N
            X = FLOAT(K)
            SUM = SUM + 1.0/X
  100 CONTINUE
      WRITE(6, 20) N, SUM
   20 FORMAT(1X, I5, 3X, F10.3)
      STOP
      END
```

(*b*) El programa es el mismo de (*a*), excepto que la proposición DO se debe reemplazar por

```
      DO 100 K =  1, N, 2
```

puesto que solamente se utilizan valores impares de K.

(*c*) El programa es el mismo de (*a*), excepto que la sexta línea se debe reemplazar por

```
      SUM = SUM + (-1.0)**(K + 1)/X
```

puesto que los signos están alternados.

5.8 Suponga que se deposita un capital de $2000.00 en una cuenta de ahorros en 1977 y suponga que el banco paga anualmente 6% de interés compuesto. Escriba un programa que imprima el AÑO y el CAPIT de la cuenta hasta 1995. (Compárese con el problema 4.11.)

Recordemos primero que cada año el CAPIT se incrementa en un 6%:

$$\text{CAPIT} \leftarrow \text{CAPIT} + 0.06 * \text{CAPIT}$$

(La fórmula general es

$$\text{CAPIT} \leftarrow \text{CAPIT} + \text{TASA} * \text{CAPIT} = \text{CAPIT}\,(1 + \text{TASA})$$

donde TASA es la tasa de interés.) El programa es el siguiente:

```
      INTEGER AÑO
      WRITE(6, 10)
   10 FORMAT('1', 6X, 'AÑO', 6X, 'CAPITAL'//)
      CAPIT = 2000.00
      AÑO = 1977
      WRITE(6, 20) AÑO, CAPIT
   20 FORMAT(1X, I10, 3X, '$', F8.2)
      DO 99 AÑO = 1978, 1995
            CAPIT = CAPIT + 0.06*CAPIT
            WRITE(6, 20) AÑO, CAPIT
   99 CONTINUE
      STOP
      END
```

(Obsérvese que utilizamos un ciclo DO en el cual el índice AÑO se inicializa en 1978, puesto que es el primer año en que se obtiene un interés.)

5.9 Suponga que un conjunto de datos tales como $x_1, x_2, \ldots, x_n$ se perfora en tarjetas, a razón de un número por tarjeta y que el paquete contiene una tarjeta de encabezamiento.

(*a*) Escriba un segmento de programa que calcule las sumas:

$$\text{SUM} = x_1 + x_2 + \cdots + x_n = \sum_{i=1}^{n} x_i$$

$$\text{SUMCUA} = x_1^2 + x_2^2 + \cdots + x_n^2 = \sum_{i=1}^{n} x_i^2$$

[El símbolo de sumatoria Σ se estudia en la sección 8.9(*c*).]

(*b*) La media, la varianza y la desviación estándar de $x_1, x_2, \ldots, x_n$ se definen como

$$m = \text{media} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\text{varianza} = \frac{\sum_{i=1}^{n} (x_i - m)^2}{n}$$

$$\text{desviación estándar} = \sqrt{\text{varianza}}$$

Sin embargo, la varianza también se puede escribir

$$\text{varianza} = \frac{\sum_{i=1}^{n} x_i^2}{n} - m^2$$

Escriba un programa que calcule la media, la varianza y la desviación estándar de los datos.

(*a*) Primero, consideramos SUM = 0.0 y SUMCUA = 0.0. Después de leer N, utilizamos un ciclo DO para leer los valores de X y también para sumar X a SUM y X^2 a SUMCUA. El segmento de programa es el siguiente:

```
      SUM = 0.0
      SUMCUA = 0.0
      READ(5, 10) N
10    FORMAT(I5)
      DO 100 I = 1, N
            READ(5, 20) X
20          FORMAT(F10.2)
            SUM = SUM + X
          SUMCUA = SUMCUA + X**2
100   CONTINUE
```

(*b*) La segunda fórmula para la varianza nos permite calcular la varianza al mismo tiempo que la media (sin usar variables subindicadas). Utilizamos en nuestro programa el segmento de programa visto anteriormente:

```
      REAL MEDIA
      SUM = 0.0
      SUMCUA = 0.0
      READ(5, 10) N
10    FORMAT(I5)
      DO 100 I = 1, N
            READ(5, 20) X
20          FORMAT(F10.2)
            SUM = SUM + X
            SUMCUA  = SUMCUA + X**2
100   CONTINUE
      XN = FLOAT(N)
      MEDIA = SUM/XN
```

```
      VAR = SUMCUA/XN - MEDIA**2
      SD = SQRT(VAR)
      WRITE(6, 30) MEDIA, VAR, SD
30    FORMAT(1X, 3(F10.2, 2X))
      STOP
      END
```

5.10 Una manera de calcular la depreciación es por medio del método de la suma de dígitos. Por ejemplo, supongamos que un automóvil con costo de $4500.00 se deprecia en cinco años. En este caso, la suma de los dígitos de los años es:

$$SUM = 1 + 2 + 3 + 4 + 5 = 15$$

De acuerdo con este método, la depreciación del primer año es 5/15 de $4500, la del segundo año 4/15, la del tercer año 3/15 y así sucesivamente. Estos se tabulan de la manera siguiente:

Año	Depreciación
1	1500.00
2	1200.00
3	900.00
4	600.00
5	300.00

Escriba un programa que lea un número real positivo COST y un entero positivo N e imprima la depreciación de COST en N años, utilizando el método de la suma de dígitos; la salida debe aparecer en la misma forma que en la tabla anterior.

Haremos la fórmula

$$SUM = 1 + 2 + 3 + \cdots + n = n(n + 1)/2$$

Obsérvese que n/SUM de COST es la depreciación del primer año, $(n-1)$/SUM la del segundo año y así sucesivamente. En particular, $(n - K + 1)$/SUM de COST es la depreciación del K-ésimo año. En el programa, utilizamos un ciclo DO con índice K = 1, 2,..., N para calcular la depreciación a partir de

```
DEP = FLOAT(N - K + 1)/SUM*COST
```

El programa es el siguiente:

```
C
C     PROGRAMA PARA DEPRECIACION UTILIZANDO SUMA DE DIGITOS
C
      READ(5, 10) COST, N
10    FORMAT(F15.2, I5)
      SUM = N*(N + 1)/2
      WRITE(6, 20)
20    FORMAT('1', 5X, 'AÑO', 5X, 'DEPRECIACION')
      DO 100 K = 1, N
            DEP = FLOAT(N - K + 1)/SUM*COST
            WRITE(6, 30) K, DEP
30          FORMAT(6X, I3, 6X, F10.2)
100   CONTINUE
      STOP
      END
```

5.11 (*a*) Considere el polinomio cuadrático $y = 2x^2 - 3x - 5$. Escriba un programa Fortran que encuentre el valor de y para valores de x desde -4 hasta 4 con incrmentos de 0.5.
(*b*) Considere el polinomio

$$z = x^3 - 3xy^2 + 2xy + y - 2y^3$$

Escriba un programa Fortran, utilizando ciclos DO anidados, que encuentre el valor de z para valores de x y y desde -4 hasta 4 con incrementos de 0.5.

(*a*) Hay 17 valores de x, de manera que utilizaremos un ciclo DO con índice I = 1, 2, ..., 17. La relación entre el índice I y x *es la siguiente:*

x:	−4,	−3.5,	−3,	−2.5,	...,	3.5,	4
I:	1,	2,	3,	4,	...,	16,	17

Obsérvese que x se puede obtener de I a partir de

$$x = -4 + 0.5(I - 1)$$

El programa es como sigue:

```
C       PROGRAMA PARA UN POLINOMIO CUADRATICO
        DO 100 I = 1, 17
              X = -4.0 + 0.5*FLOAT(I - 1)
              Y = 2.0*X**2 - 3.0*X - 5.0
              WRITE(6, 10) X, Y
   10         FORMAT(1X, F10.3, 3X, F10.3)
  100   CONTINUE
        STOP
        END
```

(*b*) El programa es similar al de (*a*), excepto que ahora tendremos un ciclo DO para x y otro para y:

```
C       PROGRAMA PARA UN POLINOMIO
        DO 200 I = 1, 17
              X = -4.0 + 0.5*FLOAT(I - 1)
              DO 100 J = 1, 17
                    Y = -4.0 + 0.5*FLOAT(J - 1)
                    Z = X**3 - 3.0*X*Y**2 + 2.0*X*Y + Y - 2.0*Y**3
                    WRITE(6, 10) X, Y, Z
   10               FORMAT(1X, 3(3X, F10.2))
  100         CONTINUE
  200   CONTINUE
        STOP
        END
```

5.12 El gráfico de la ecuación $x^2 + y^2 = 50$ es un círculo C con centro en el origen y radio $\sqrt{50}$.

(*a*) Determine el número de puntos con coordenadas enteras positivas que estén dentro del círculo.
(*b*) Determine el número de puntos con coordenadas enteras que estén dentro del círculo.

(*a*) Nótese primero que ni x ni y pueden exceder 7, ya que el radio es $\sqrt{50}$. Utilizamos ciclos DO anidados, uno para x y otro para y. Comenzamos también con un contador K inicializado en 0 y no contamos aquellos puntos (x, y) para los cuales $x^2 + y^2 \geq 50$. Sigue el programa. Nótese que x, y se declaran como variables enteras.

```
C     PROGRAMA PARA CONTAR LOS PUNTOS DENTRO DE UN CIRCULO
      INTEGER X, Y
      K = 0
      DO 100 X = 1, 7
            DO 200 Y = 1, 7
                  IF(X**2 + Y**2.GE.50) GO TO 200
                  K = K + 1
  200       CONTINUE
  100 CONTINUE
      WRITE(6, 10) K
   10 FORMAT(1X, 'EL NUMERO DE PUNTOS ES', 2X, I5)
      STOP
      END
```

(*b*) Aquí tenemos que examinar 15 valores para x y y, esto es, $-7, -6, \ldots, 7$. Por lo tanto, consideramos para x un ciclo DO con índice I = 1, 2,..., 15, donde $x = 8 - I$ y además un ciclo DO similar para y. De otra manera, el programa es semejante al de la parte (*a*).

```
      INTEGER X, Y
      K = 0
      DO 100 I = 1, 15
            X = 8 - I
            DO 200 J = 1, 15
                  Y = 8 - J
                  IF(X**2 + Y**2.GE.50) GO TO 200
                  K = K + 1
  200       CONTINUE
  100 CONTINUE
      WRITE(6, 10) K
   10 FORMAT(1X, 'EL NUMERO DE PUNTOS ES', 2X, I5)
      STOP
      END
```

5.13 Encuentre todos los números primos de tres dígitos; esto es, encuentre todos los números primos entre 100 y 999. (Compárese con el ejemplo 5.6.)

Ninguno de estos números primos puede ser par; por lo tanto, consideramos solamente los enteros impares K entre 101 y 999 para observar si K es primo. Además, si K no es primo, entonces K debe tener un divisor impar entre 3 y K/2. De esta manera, utilizamos un ciclo DO con índice K para generar los posibles números primos K y un ciclo DO con índice M para generar los posibles divisores del K dado. El diagrama de flujo y su traducción Fortran aparece en la figura 5-15(*a*) y (*b*), respectivamente.

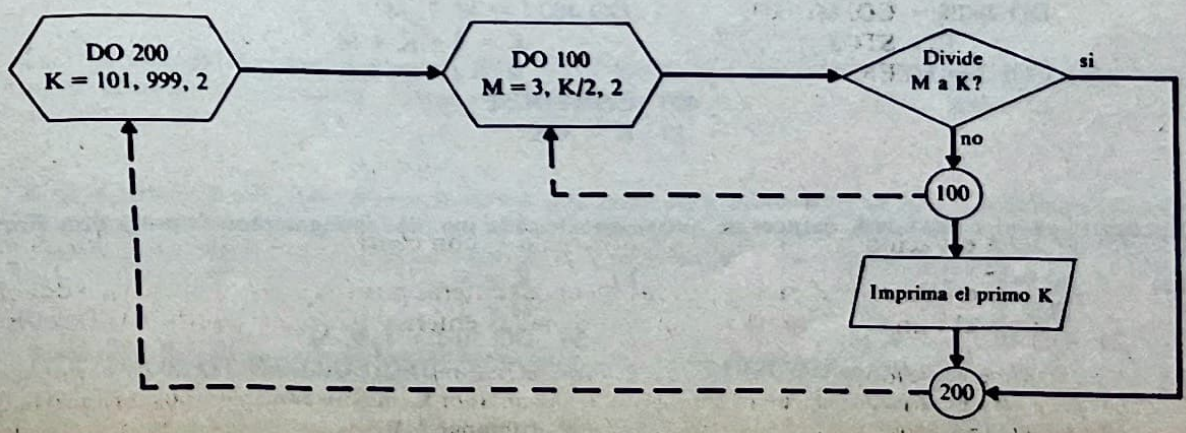


Fig. 5-15(*a*)

```
C
C       PROGRAMA PARA IMPRIMIR NUMEROS PRIMOS
C
        DO 200 K = 101, 999, 2
                KK = K/2
                DO 100 M = 3, KK, 2
                        IF(K.EQ.(K/M)*M) GO TO 200
  100           CONTINUE
                WRITE(6, 10) K
   10           FORMAT(1X, I10)
  200   CONTINUE
        STOP
        END
```

Fig. 5-15(*b*)

Problemas suplementarios

CICLOS DO

5.14 Determine el número de veces y los valores del índice para los cuales se ejecuta un ciclo DO si su primera proposición es:

(*a*) DO 100 L = 2, 15, 3 (*c*) DO 300 I = 6, 9, 5
(*b*) DO 200 JOHN = 5, 11 (*d*) DO 400 J = 8, 5, 4

5.15 Encuentre los errores, si los hay, en cada proposición DO:

(*a*) DO 700, LAMB = 1, 14, L, (*c*) DO 900 K = I, J, K,
(*b*) DO 800 J = 7, M**2, 2 (*d*) DO 1000 LONG = K234, K123, K345

5.16 Encuentre el valor de K después de que se ejecute cada uno de los segmentos de programa Fortran:

(*a*)
```
        K = 3
        DO 100 J = 3, 7, 3
                K = K + J
  100   CONTINUE
        K = 3*K
```

(*b*)
```
        K = 3
        M = 2
        DO 200 J = 3, 7, M
                K = K + J
  200   CONTINUE
        K = 3*K
```

(*c*)
```
        K = 3
        M = 2
        DO 300 J = 3, 7, M
                M = M + K
  300   CONTINUE
        K = 3*K
```

(*d*)
```
        K = 3
        M = 2
        DO 400 J = M, 7, M
                K = J + K + M
                IF(K.GT.9) GO TO 10
  400   CONTINUE
   10   K = 3*K
```

5.17 Encuentre el valor final de K después de que se ejecute cada uno de los segmentos de programa Fortran:

(*a*)
```
        K = 3
        M = 2
   20   DO 10 J = 4, 9, M
                IF(J.EQ.6) GO TO 10
                K = K + 2*J
   10   CONTINUE
        K = 3*K
```

(*b*)
```
        K = 3
        M = 2
   20   DO 10 J = 4, 9, M
                IF(J.EQ.6) GO TO 20
                K = K + J**2
   10   CONTINUE
        K = 3*K
```

5.18 Suponga que cada flecha curva representa una transferencia de control. Encuentre los errores en cada programa.

(*a*)
```
C     PRIMER PROGRAMA
      ...................
      ...................
      DO 100 K = 1, 5
      ...................
      ...................
      ...................
100   CONTINUE
      ...................
      ...................
      END
```
a
b

(*b*)
```
C     SEGUNDO PROGRAMA
      ...................
      ...................
      DO 100 K = 1, 5
      ...................
      ...................
      ...................
      DO 100 L = 1, 5
      ...................
      ...................
      ...................
100   CONTINUE
      ...................
      ...................
      END
```
a
b
c

(*c*)
```
C     TERCER PROGRAMA
      ...................
      DO 100 K = 1, 5
      ...................
      ...................
      DO 200 L = 1, 5
      ...................
      ...................
100   CONTINUE
200   CONTINUE
      ...................
      END
```
a
b

(*d*)
```
C     CUARTO PROGRAMA
      ...................
      DO 100 K = 1, 5
      ...................
      DO 200 L = 1, 5
      ...................
200   CONTINUE
      ...................
      DO 300 M = 1, 5
      ...................
300   CONTINUE
      ...................
100   CONTINUE
      ...................
      END
```
a
b
c
d
e

PROGRAMAS

(Muchos de estos problemas son los mismos del capítulo 4. Sin embargo, ahora ellos se resolverán utilizando un ciclo DO en lugar de un contador.)

5.19 Escriba un programa para imprimir un número impar de dos dígitos N, su cuadrado N^2 y su cubo N^3, tal que cada N diferente aparezca en líneas distintas. (Compárese con el problema 4.9.)

5.20 Escriba un segmento de programa Fortran que imprima el número 20 veinte veces, el número 19 diecinueve veces, el número 18 dieciocho veces, y así sucesivamente.

5.21 Suponga que N y K son enteros positivos que están en la memoria, con $K < N$. Escriba un segmento de programa que utilice una proposición DO para imprimir N, N—1, N—2 y así sucesivamente, en forma decreciente, hasta cuando se imprima el valor de K. [Sugerencia: Véase ejemplo 5.5(*b*).]

5.22 Dados A, B y N, escriba un segmento de programa Fortran para calcular

$$\frac{1}{A}+\frac{1}{A+B}+\frac{1}{A+2B}+\frac{1}{A+3B}+\cdots+\frac{1}{A+NB}$$

5.23 Escriba un programa que lea un entero positivo $N \geq 10$ y calcule, con cinco lugares decimales, el producto

$$\frac{1}{1^2} \cdot \frac{3}{2^2} \cdot \frac{5}{3^2} \cdot \ldots \cdot \frac{2N-1}{N^2}$$

5.24 Escriba un programa que imprima todos los enteros positivos impares menores que 100, omitiendo aquellos enteros divisibles por 7:

1, 3, 5, 9, 11, 15, 17, 19, 23, . . . , 97, 99

5.25 Suponga que se leen como datos el CAPIT de depósito, la TASA de interés (compuesto y anual) y el número N de años. Escriba un programa que imprima el capital de la cuenta cada año durante un período de N años. (Compárese con el problema 5.8.)

5.26 Suponga que *cada año* se depositan \$500.00 en una cuenta de ahorros que produce anualmente 7% de interés compuesto. Escriba un programa que imprima el capital de la cuenta cada año durante un período de 10 años.

5.27 Suponga que *cada dos años* se depositan \$500.00 en una cuenta de ahorros que produce anualmente 7% de interés compuesto. Escriba un programa que imprima el capital de la cuenta cada año durante un período de 10 años.

5.28 Un hombre recibe un préstamo de \$300.00 de un banco, a una tasa de interés igual a 1.5% mensual. Amortiza \$25.00 al final de cada mes. (De esta manera, adeuda al final del primer mes

$$\begin{aligned} \text{CAPIT} + \text{INTERES} - \text{AMORTIZACION} &= 300 + (1.5\%)(300) - 25 \\ &= 300 + 4.50 - 25 = 279.50 \end{aligned}$$

pesos.)

(*a*) Escriba un programa Fortran que imprima el capital que él debe cada mes durante un período de un año.
(*b*) Escriba un segmento de programa Fortran que encuentre el número de meses en los que él debe hacer pagos sobre el préstamo y el valor de su último pago.

5.29 Encuentre el número de puntos con coordenadas enteras que están dentro de la elipse $2x^2 + 3y^2 = 100$.

5.30 Tres enteros positivos a, b, c, donde $a < b < c$, forman una terna pitagórica si $a^2 + b^2 = c^2$. Por ejemplo, 3, 4 y 5 forman una terna pitagórica, puesto que $3^2 + 4^2 = 5^2$. Escriba un programa que encuentre todas las ternas pitagóricas a, b, c donde $a, b < 25$.

5.31 Suponga que cada tarjeta de un paquete contiene un número real. Se adiciona una tarjeta de encabezamiento que contiene el número N de tarjetas en el paquete. Encuentre los valores mayor y menor.

5.32 Suponga que se perforan en tarjetas, 25 enteros positivos, a razón de uno por tarjeta. Escriba un programa para encontrar el segundo entero mayor.

5.33 Suponga que en un paquete de tarjetas cada una contiene un entero positivo. Escriba un programa que encuentre el mayor entero par en el paquete, o que imprima NO HAY ENTERO PAR si todos los enteros son impares. Suponga que el paquete tiene una tarjeta de encabezamiento con el número N de tarjetas en el paquete.

5.34 Suponga que un paquete de tarjetas contiene un entero positivo por tarjeta. Escriba un programa para encontrar el número de enteros pares y el número de enteros impares. (Suponga que el paquete tiene una tarjeta de encabezamiento.)

5.35 La constante π se puede aproximar a partir de:

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots$$

Escriba un programa que:

(*a*) Sume los 1000 primeros términos.
(*b*) Sume los 1000 primeros términos en sentido inverso.

(Pregunta: ¿Hay alguna diferencia?)

5.36 Considere la ecuación $y = x^2 - 4x + 6$. Escriba un programa Fortran que encuentre el valor de y para valores de x:

(*a*) Desde -5 hasta 5 con incrementos de 0.1.
(*b*) Desde I hasta J con incrementos de D = 1/N, donde I, J y N se perforan en una tarjeta e I < J.

(Cada valor de x y su correspondiente valor de y deben aparecer en líneas diferentes.)

5.37 Considere la ecuación $z = x^2 - 2xy + 3y^2 - 8x + 3y - 8$. Escriba un programa que encuentre el valor de z para valores de x y y, donde el intervalo de x y y va desde -3 hasta 3 con incrementos de 0.2.

5.38 Modifique el problema anterior, de tal manera que el programa determine también el máximo valor de z.

5.39 El seno de x (en radianes) se puede evaluar aproximadamente si se suman los N primeros términos de la serie

$$\operatorname{sen} x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

(*a*) Escriba un programa que calcule el seno de 3 radianes a partir de la serie anterior para N = 1, 2, 3, ..., 8. Compare cada respuesta con la suministrada por la función de biblioteca SIN (X).
(*b*) Escriba un programa que lea un entero positivo N y un valor real x y que evalúe sen x, utilizando la serie anterior.
(*c*) Escriba un programa que lea un valor real x y evalúe sen x, sumando los términos sucesivos de la serie anterior, hasta cuando el último término sumado tenga un valor absoluto menor que 10^{-15}.

Respuestas a problemas suplementarios seleccionados

5.14 (*a*) Cincoveces: 2, 5, 8, 11, 14 (*b*) Sieteveces: 5, 6, 7, 8, 9, 10, 11 (*c*) Unavez: 6 (*d*) Unavez: 8

5.15 (*a*) No debe haber coma después de 700 ni después de L.
(*b*) M**2 no se puede utilizar como parámetro.
(*c*) No debe haber coma después de K. La variable índice K no puede ser uno de los parámetros.
(*d*) No hay error.

5.16 (*a*) 36 (*b*) 54 (*c*) El error está en que se altera el valor del parámetro M. (*d*) 39

5.17 (*a*) 81 (*b*) El computador nunca finaliza el segmento. [Véase el problema resuelto 5.4(*b*).]

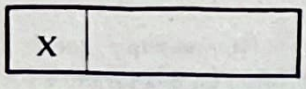
5.18 (*a*) La transferencia a no está permitida.
(*b*) Las transferencias a y c no están permitidas.
(*c*) Las proposiciones CONTINUE deben intercambiarse.
(*d*) Las transferencias a y d no son permitidas.

Capítulo 6

Arreglos, variables subindicadas

6.1 INTRODUCCION

Las variables que hemos utilizado hasta ahora, se llaman *variables no subindicadas* o *variables escalares.* Cada una de estas variables escalares representa una celda de memoria en la cual se puede almacenar un valor único; es decir, una variable llamada X podría considerarse como una caja:



Muchas veces, quisiéramos utilizar el mismo nombre para referirnos a una lista de valores que tienen los mismos atributos comunes, por ejemplo, el número de identificación de los estudiantes de la clase de programación de computadores, las notas de un examen final y así sucesivamente. Esto se puede hacer por medio de la utilización de arreglos o variables subindicadas.

En matemáticas, se usa un símbolo con diferente subíndice para indicar los elementos en un grupo dado. Por ejemplo, $s_1, s_2, \ldots, s_{25}$ pueden denotar las notas de una clase de 25 estudiantes. Utilizando la notación con subíndices, se puede escribir:

$$\sum_{i=1}^{25} s_i = s_1 + s_2 + \cdots + s_{25}$$

para la suma de las notas, y

$$\left(\sum_{i=1}^{25} s_i\right)/25$$

para el promedio de clase. En otras palabras, el uso de subíndices es absolutamente necesario en el desarrollo de una notación concisa, para la manipulación algebraica de los números involucrados.

Los subíndices se utilizan también en Fortran, pero tienen una apariencia diferente. Puesto que cada proposición Fortran debe ser perforada (o escrita) sobre una línea, se emplean paréntesis para encerrar los subíndices, y escribiríamos S(1), S(2),..., S(25) en lugar de $S_1, S_2, \ldots, S_{25}$. Los cómputos cuyos elementos se especifican por medio de un subíndice se llaman en Fortran *arreglos lineales* o *unidimensionales.* Conjuntos análogos cuyos elementos se especifican por medio de dos o más variables, se llaman *arreglos multidimensionales.*

6.2 ARREGLOS UNIDIMENSIONALES

Un arreglo lineal o unidimensional con N elementos es una lista o cadena de N celdas de memoria, donde cada celda puede almacenar un valor. Estas celdas tienen el mismo nombre y son contiguas una a otra. Cualquier celda particular en este arreglo unidimensional se identifica por un entero K que indica su posición en el arreglo. (Luego $1 \leq K \leq N$.)

Por ejemplo, supongamos que ID es un arreglo unidimensional con 50 elementos. Entonces, puede imaginarse ID como una cadena de 50 cajas, como la que se presenta en la figura 6-1. Cada celda en el arreglo ID se referencia por su nombre y su posición, llamada *subíndice*. Específicamente, ID(1) denota el primer elemento del arreglo, ID(2) denota el segundo elemento del arreglo, y así sucesivamente, Más aún, si K está definido y $1 \leq K \leq 50$, entonces ID(K) se refiere al elemento número K del arreglo.

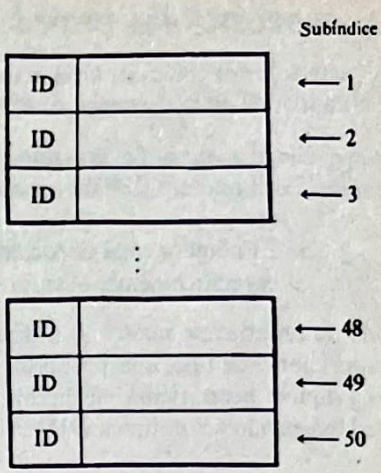


Fig. 6-1

Las reglas utilizadas para nombrar variables, también se aplican a los arreglos; es decir, un nombre de arreglo está constituido por uno a seis caracteres alfanuméricos, siendo el primer carácter alfabético. Además, si el nombre del arreglo empieza con una de las letras I, J, K, L, M o N, entonces el arreglo sólo puede contener valores enteros. De la misma forma, si no empieza con las letras anteriores, sólo puede contener valores reales. Aunque esas convenciones pueden ser superadas por una proposición de tipo (que se estudiará en la sección 6.3), los elementos en cualquier arreglo deben ser siempre del *mismo* tipo.

Si bien un elemento de un arreglo está determinado por el nombre del arreglo seguido de los subíndices apropiados (encerrados entre paréntesis), cada elemento de un arreglo puede aparecer en expresiones aritméticas, como variables escalares. Por ejemplo, las siguientes expresiones tienen una sintaxis correcta:

```
X = AMOUNT(5) + Q
PAY(K) = HOUR(K)*40.0
TAX(6) = 25.75
```

Por otra parte, no podemos utilizar sólo los nombres de los arreglos en proposiciones aritméticas. Por ejemplo, si GRADE es un arreglo y SUM es una variable ordinaria, proposiciones tales como:

```
SUM = GRADE + SUM
GRADE = SUM + 10.0
GRADE = GRADE + 20.0
```

no se permiten. Esto, por cuanto el nombre GRADE se refiere a un arreglo entero, mientras que su aritmética envuelve localizaciones individuales de memoria.

Podemos también referirnos a elementos de un arreglo en proposiciones de entrada/salida. Por ejemplo,

```
READ(5, 10) X, TAX(3)
```

instruirá al computador para que lea dos números. El primer número está almacenado en la localización X y el segundo está almacenado en el tercer elemento del arreglo llamado TAX. Imprimimos desde la memoria los números de identificación ID(1), ID(2),..., ID(K), uno por línea, usando el siguiente segmento de programa.

```
      DO 100 I = 1, K
            WRITE(6, 20) ID(I)
 20         FORMAT(11X, I10)
100   CONTINUE
```

Aunque los nombres de los arreglos no pueden aparecer solos en proposiciones aritméticas, pueden aparecer solos en proposiciones de entrada/salida. Esto se discutirá posteriormente, en este mismo capítulo.

Nota. El término *variable subindicada* se emplea en diferentes textos, con significados diferentes. Algunos textos emplean la variable subindicada para significar el arreglo en sí, mientras que para otros, ella se refiere sólo a un elemento del arreglo. Así, en algunos textos dirán que el arreglo anterior ID es una variable subindicada, mientras que otros textos dirán que los elementos ID(3), ID(4), etc., son variables subindicadas. Muchos manuales IBM sobre Fortran no emplean el término variable subindicada. En cambio, todos los textos usan el término variable no subindicada para significar la misma cosa, variables escalares ordinarias.

6.3 PROPOSICIONES DIMENSION

antes de que cualquier arreglo lineal o multidimensional sea empleado en un programa, se debe proveer al compilador de la siguiente información:

1. El nombre del arreglo.
2. El número de subíndices en el arreglo (es decir, si el arreglo es unidimensional, bidimensional, etc.)
3. El número total de localizaciones de memoria que se necesitan para su almacenamiento o, más específicamente, el valor máximo de cada subíndice,

Todo lo anterior se realiza al utilizar la proposición de tipo DIMENSION. (Como ocurre en todas las proposiciones de tipo, una proposición DIMENSION no es ejecutable y debe colocarse en el programa antes de cualquier proposición ejecutable.)

Una proposición típica DIMENSION es como sigue:

```
DIMENSION AMOUNT(100)
```

Esta proposición instruye al compilador de que AMOUNT es el *nombre* de un *arreglo real lineal* y que se deben separar 100 localizaciones de memoria para su almacenamiento.

En lugar de listar los arreglos separadamente, se pueden declarar varios arreglos con una sola proposición DIMENSION. Por ejemplo,

```
DIMENSION ID(50), TAX(150)
```

declara que ID es un arreglo entero lineal con 50 elementos y que TAX es un arreglo real lineal con 150 elementos. (La forma utilizada para arreglos multidimensionales es similar y será estudiada posteriormente.) El orden de la lista de arreglos no interesa, siempre y cuando cada variable que deba ser subindicada sea incluida en la proposición DIMENSION. Obsérvese también que no hay coma después de la palabra DIMENSION, pero que los diferentes arreglos están separados por coma en la proposición DIMENSION.

Por lo general, no se permiten subíndices que sean cero o negativos. Obviamente, no se podrá usar un subíndice superior al tamaño máximo especificado en la proposición DIMENSION. Por ejemplo, dada la proposición DIMENSION del ejemplo anterior, no se podrá utilizar en el programa ID(55) o TAX(182). Debe tenerse especial cuidado en esto, ya que muchos compiladores no chequean la validez de tales subíndices. Por lo tanto, errores de este tipo pueden no ser detectados y ser ejecutados, llegando a resultados erróneos.

Anotamos que la proposición DIMENSION separa el número máximo de celdas de memoria que requiere un arreglo, aunque el programador utilice sólo una parte de este arreglo. Por lo general, este sobredimensionamiento puede ser necesario, pero debe hacerse sobre bases realistas, en términos del problema en sí y de sus requerimientos, Por ejemplo, bien podemos no conocer el número exacto de estudiantes que presentaron un examen (es decir, algunos estudiantes estuvieron ausentes), pero podemos utilizar

```
DIMENSION TEST(60)
```

suponiendo condiciones normales de clase. Hacemos énfasis en el hecho de que el uso irreflexivo de arreglos grandes será costoso en razón del tiempo de procesamiento, y de la sobrecarga de las condiciones de computación.

Nota. Hacemos énfasis en que se separan las celdas de memoria para los arreglos durante la compilación del programa y no durante su ejecución; entonces, el subíndice especificado en cualquier proposición DIMENSION debe ser una *constante entera* sin signo. (La única excepción puede presentarse en los SUBPROGRAMAS; véase el capítulo 7.) En consecuencia, son inaceptables las proposiciones de la forma

```
DIMENSION AMOUNT(K)
```

ó

```
N = 75
DIMENSION SCORE(N)
```

```
6
      READ(5, 10) I
      DIMENSION GRADE(I)
```

Hay un último punto que deseamos tratar en esta sección. Supongamos que deseamos que el arreglo SCORE, contenga a lo sumo 50 enteros (no números reales). Esto se puede hacer utilizando las proposiciones de tipo

```
      INTEGER SCORE
      DIMENSION SCORE(50)
```

Sin embargo, se puede usar alternativamente la proposición

```
      INTEGER SCORE(50)
```

que también provee la información necesaria para el compilador. De modo similar,

```
      REAL INVEST(150)
```

definirá a INVEST como el nombre de un arreglo lineal con 150 valores reales. Como es usual, puede aparecer más de un arreglo en tales proposiciones, que deberá estar separado con comas.

6.4 EXPRESIONES ARITMETICAS PARA SUBINDICES

Una de las principales ventajas del uso de arreglos es que los subíndices pueden a su vez ser variables. En efecto, los subíndices pueden ser expresiones aritméticas simples limitadas a una de las siguientes formas:

n, K, K + n, n*K, n*K ± m

donde *n* y *m* son constantes enteras y K es una variable entera. Por ejemplo, tienen sintaxis correcta las expresiones siguientes:

A(8), A(K), S(K – 2), A(3*K), A(4*K – 3)

Por otra parte, las siguientes expresiones no son aceptables:

A(–4)	(No se permiten subíndices negativos.)
A(0)	(No se permite el subíndice cero.)
A(K*2)	(Se debería escribir A(2*K).)
A(K**2)	(El subíndice no tiene una de las formas mencionadas anteriormente.)
A(.TRUE.)	(El subíndice no puede asumir valores lógicos.)
A(3*M(2))	(Pueden aparecer solamente variables escalares en expresiones subindicadas.)

Hacemos notar que algunos grandes computadores, modernos, permiten cualquier valor de expresiones aritméticas enteras como subíndices; en tal caso, estas reglas deben ser modificadas, como corresponde.

Aunque hemos discutido sólo arreglos unidimensionales, las reglas anteriores sobre subíndices también se aplican a arreglos multidimensionales.

6.5 EJEMPLOS QUE UTILIZAN ARREGLOS

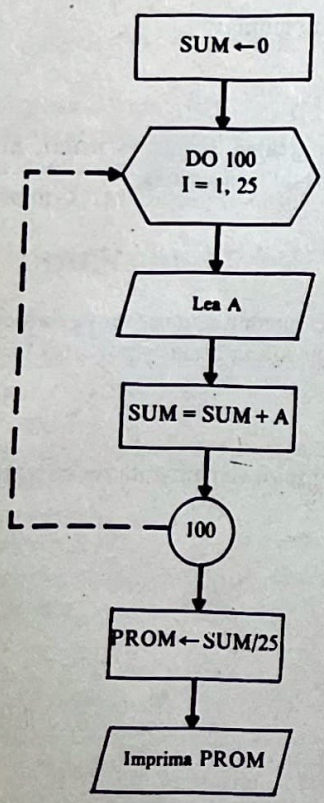
Supongamos que se hace un examen para una clase de no más de 25 estudiantes y que las notas están perforadas sobre tarjetas, una por tarjeta. Deseamos escribir un programa Fortran para:

1. Encontrar el promedio de la clase, y
2. Contar el número de estudiantes cuyas notas están por debajo del promedio.

Primero, estudiemos el caso más simple en donde, exactamente, 25 estudiantes tomaron el examen. (Para encontrar el promedio de clase, debemos primero encontrar el gran total de todas las notas; entonces, recordemos lo visto en la sección 4.10 para encontrar la suma de varios números.)

Una forma de atacar el problema es utilizar 25 nombres distintos, digamos A1, A2,..., A25 para las 25 notas. Esto nos conduciría a proposiciones muy grandes que estarían sujetas a error. (Véase página 94 .) Además, se requerirían 25 proposiciones IF para comparar cada nota con el promedio de clase. Más aún, el programa se volvería completamente inmanejable si, digamos, se aumentaran las notas a 100 o más. Luego, miraremos otra forma de resolver el problema.

El promedio de clase puede ser calculado por el diagrama de flujo y el segmento de programa de la figura 6-2. (Compárese con la figura 4-17.) Sin embargo, no hay forma de comparar cualquier nota con el promedio de clase, puesto que las notas no están en la memoria del computador. Luego, no se puede encontrar el número de estudiantes cuyas notas están por debajo del promedio de clase.

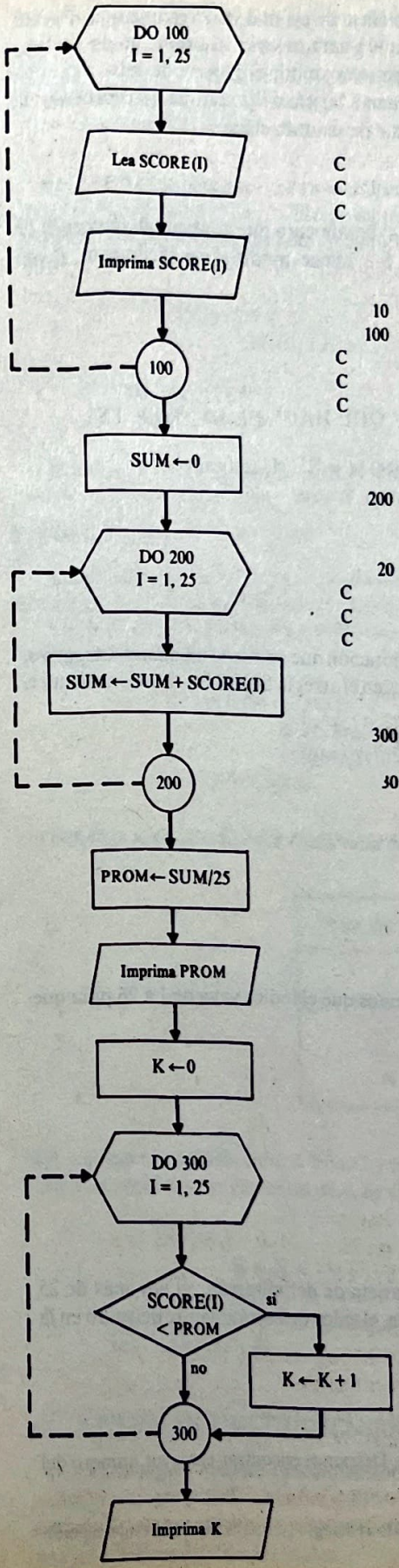



```
      SUM = 0.0
      DO 100 I = 1, 25
             READ(5, 10) A
10           FORMAT(F10.1)
             SUM = SUM + A
100   CONTINUE
      PROM = SUM/25.0
      WRITE(6, 20) PROM
20    FORMAT(1X, 'EL PROMEDIO DE LA CLASE ES', 1X, F4.1)
```

Fig. 6-2

El uso de variables subindicadas es absolutamente necesario si deseamos utilizar el esquema anterior y, además, calcular el número de estudiantes que tienen notas por debajo del promedio de la clase. Específicamente, almacenaremos las notas en un arreglo lineal llamado SCORE con 25 elementos. El diagrama de flujo y su equivalente Fortran aparecen en la figura 6-3.

Nota. La primera proposición WRITE en la figura 6-3, sirve para dos propósitos. Además, de imprimir las notas, también sirve como prueba para ver si el arreglo SCORE ha sido leído correctamente. Esto se llama una *eco-prueba*. Hacemos notar que la eco-prueba es una práctica muy útil, empleada a menudo en la *depuración*, es decir, se usa para encontrar errores en el programa. Cuando los programas con complejos, a menudo se insertan muchas proposiciones intermedias WRITE para mostrar resultados intermedios; así, los errores pueden localizarse e identificarse fácilmente. Después de que todas las *depuraciones* (errores) se han encontrado, se remueven esas proposiciones WRITE. (Aunque las primeras proposiciones READ y WRITE aparecen con su propio DO, es claro que podrían haber sido combinados con el segundo DO en un solo ciclo.)




```
      DIMENSION SCORE(25)
C
C     LECTURA DE LAS NOTAS
C
      DO 100 I = 1, 25
            READ(5, 10) SCORE(I)
            WRITE(6, 10) SCORE(I)
   10       FORMAT(F10.1)
  100 CONTINUE
C
C     CALCULO DE LA SUMA Y EL PROMEDIO
C
      SUM = 0.0
      DO 200 I = 1, 25
            SUM = SUM + SCORE(I)
  200 CONTINUE
      PROM = SUM/25.0
      WRITE(6, 20) PROM
   20 FORMAT(1X, 'EL PROMEDIO DE CLASE ES'
C
C     CONTEO DEL NUMERO DE NOTAS POR DEBAJO DEL PROMEDIO
C
      K = 0
      DO 300 I = 1, 25
            IF(SCORE(I).LT.AVE) K = K + 1
  300 CONTINUE
      WRITE(6, 30) K
   30 FORMAT(1X,  'EL NUMERO DE NOTAS POR DEBAJO DEL
                  PROMEDIO ES', 1X, I3)
      STOP
      END
```

Fig. 6-3

Consideremos ahora el problema original, que era para una clase de no más de 25 estudiantes. Puesto que se necesita el número N de estudiantes que tomaron el examen para calcular el promedio de la clase, entonces se debe definir N en el programa. Consideremos dos casos, uno donde el paquete de datos tiene una tarjeta de encabezamiento y otro donde el paquete de datos tenga una tarjeta delimitadora. (Recordemos la sección 4.11, para un estudio de las tarjetas de encabezamiento y de delimitación.)

Caso A

Supongamos que el paquete de datos tiene una tarjeta de encabezamiento que contiene el número N de estudiantes que hacen el examen. El programa de la figura 6-3 puede modificarse fácilmente, como sigue: Insertar después de la proposición DIMENSION:

```
      READ(5, 40) N
   40 FORMAT(I3)
      WRITE(6, 50) N
   50 FORMAT('1', 'EL NUMERO DE ESTUDIANTES QUE HACE EL EXAMEN ES',
```

cambiar 25 por N en las tres proposiciones DO y reemplazar PROM = SUM/25.0 por

```
      PROM = SUM/FLOAT(N)
```

De otro modo, el programa es idéntico.

Caso B

Supongamos que el paquete de datos tiene una tarjeta de delimitación que contiene un número negativo. El siguiente segmento de programa Fortran almacena los resultados en el arreglo SCORE y también cuenta el número N de estudiantes que tomaron el examen:

```
      I = 0
  111 READ(5, 60) A
   60 FORMAT(F10.1)
      IF(A.LT.0.0) GO TO 222
      I = I + 1
      SCORE(I) = A
      GO TO 111
  222 N = I
```

Esto también se puede hacer mediante el uso de un ciclo DO. Hacemos que el índice vaya de 1 a 26 para que cubra 25 resultados y la tarjeta de delimitación:

```
      DO 500 I = 1, 26
            READ(5, 60) A
   60       FORMAT(F10.1)
            IF(A.LT.0.0) GO TO 222
            SCORE(I) = A
  500 CONTINUE
  222 N = I - 1
```

Obsérvese que N = I – 1 puesto que el índice I también cuenta la tarjeta de delimitación. Si hay más de 25 estudiantes, habrá una salida anormal del ciclo DO. En consecuencia, el valor de I está siempre definido en la proposición 222.

Consideremos otros ejemplos que contengan arreglos.

EJEMPLO 6.1

Supóngase que A(1), A(2),..., A(N) están en memoria y que N ≥ 2. Deseamos encontrar el mayor número del arreglo A.

(*a*) Un método es encontrar el valor del número más grande explorando el arreglo (como se discutió en la sección 4.10).

```
      XLAR = A(1)
      DO 100 I = 2, N
            IF(A(I).LE.XLAR) GO TO 100
            XLAR = A(I)
  100 CONTINUE
```

(*b*) Un segundo método, útil en clasificación de datos, es encontrar las localizaciones L del valor más grande y entonces A(L) será el valor mayor. El algoritmo es semejante al anterior, excepto que ahora *recordamos* la localización L del número mayor, en lugar del número en sí.

```
      L = 1
      DO 200 I = 2, N
            IF(A(I).LE.A(L)) GO TO 200
            L = I
  200 CONTINUE
      XLAR = A(L)
```

En el caso en que no sepamos que $N \geq 2$ (es decir, N puede ser igual a 1), entonces la proposición DO se cambiará para que el índice I vaya desde 1 hasta N. (¿Puede el lector explicar la diferencia?)

EJEMPLO 6.2

Supongamos que A(1), A(2),..., A(N) están en memoria. Usar un ciclo DO para imprimir A(N), A(N—1),..., A(2), A(1), uno por línea. [Ver ejemplo 5.5(*c*).]

Como se discutió anteriormente, el incremento de un ciclo DO no estructurado no puede ser negativo; por lo tanto, el siguiente programa no es válido:

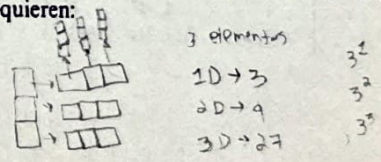
```
      DO 300 I = N, 1, -1
            WRITE(6, 10) A(I)
   10       FORMAT(10X, F10.1)
  300 CONTINUE
```

Sin embargo, a medida que el índice I varía de 1 hasta N, deseamos imprimir A(N),..., A(1). Así, pues, tenemos:

Valor del índice I	Elemento que se imprime
1	A(N)
2	A(N − 1)
3	A(N − 2)
...	...
N	A(1)

Hay una relación definida entre el índice I y el subíndice; esto es, su suma siempre es N+1. Por lo tanto, el siguiente programa imprimirá los elementos de A en el orden inverso que se requieren:

```
      DO 300 I = 1, N
            K = N + 1 - I
            WRITE(6, 10) A(K)
   10       FORMAT(10X, F10.1)
  300 CONTINUE
```



6.6 ARREGLOS MULTIDIMENSIONALES

Los arreglos lineales (unidimensionales) tienen exactamente un subíndice. Fortran también permite arreglos con dos y tres subíndices. Estos se llaman *arreglos bidimensionales* y *tridimensionales*, respectivamente. De hecho, algunos computadores grandes permiten hasta siete subíndices. En esta sección, veremos esos arreglos multidimensionales.

Matemáticamente hablando, una *matriz* se define como un *arreglo rectangular* de números, por ejemplo,

$$\begin{array}{ll} 1 \text{ fila} & \\ 2 \text{ fila} & \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \\ 3 \text{ fila} & \end{array}$$

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \text{col.} & \text{col.} & \text{col.} & \text{col.} \end{array}$$

es una matriz de tres por cuatro, es decir, una matriz con tres filas (líneas horizontales de números) y cuatro columnas (líneas verticales de números). Cada elemento de la matriz está identificado por dos enteros, sus subíndices. El primer subíndice da la *posición de la fila* del elemento y el segundo subíndice la *posición de columna*. Por ejemplo, a_{21} aparece en la segunda fila, primera columna; a_{32} aparece en la tercera fila, segunda columna y así sucesivamente. Generalmente se escribe 3×4 por "3 por 4".

Análogamente, en Fortran un arreglo bidimensional $m \times n$ es una lista de celdas de memoria (elementos del arreglo), de $m \cdot n$, en la cual un elemento particular se determina por un par de enteros K, L, donde $1 \leq K \leq m$ y $1 \leq L \leq n$. Los enteros K y L son llamados los *subíndices* del elemento; el primer subíndice K es llamado la *fila* del elemento y el segundo subíndice L, la *columna* del elemento. (Tales arreglos bidimensionales serán llamados también *arreglos matriciales.*)

Por ejemplo, supongamos que A es el nombre de un arreglo bidimensional de 3×4. Entonces, A tiene $3 \cdot 4 = 12$ elementos.

$$A(1, 1), A(1, 2), A(1, 3), A(1, 4), A(2, 1), A(2, 2), A(2, 3), \ldots, A(3, 3), A(3, 4)$$

Obsérvese que cada elementos se designa por el nombre del arreglo seguido por su par de subíndices separados por una coma y encerrados entre paréntesis. Aunque usualmente se dibuja A como un arreglo rectangular como en la figura 6-4(*a*), será representado internamente en el computador como una cadena de 12 celdas de memoria, como en la figura 6-4(*b*). Nótese que los elementos están almacenados en columnas, así que el primer subíndice (fila) varía primero y luego varía el segundo subíndice (columna). Este orden de almacenamiento se indica mediante flechas, como se ve en la figura 6-4(*a*).

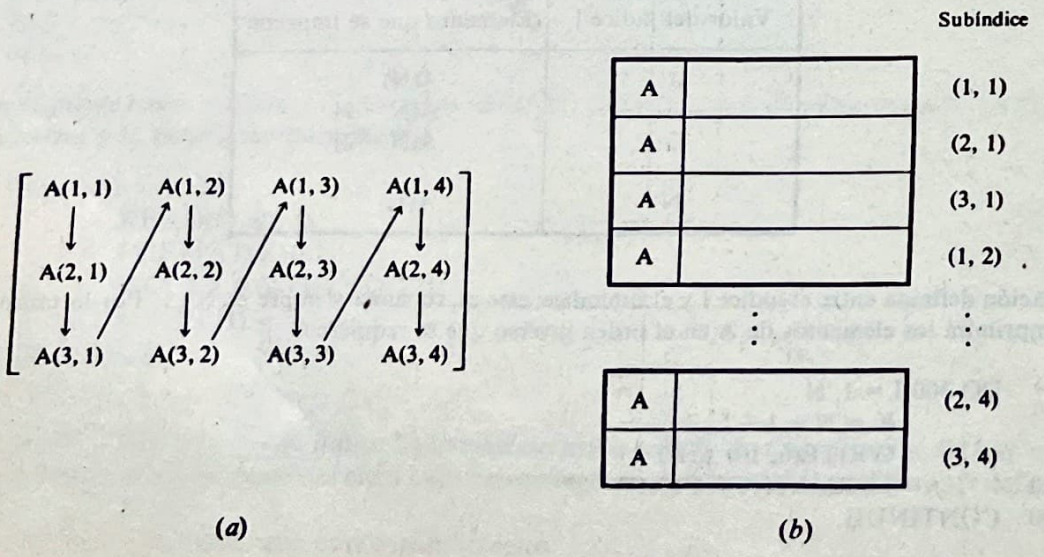


Fig. 6-4

Análogamente, un arreglo tridimensional $m \times r \times s$ es una lista de $m \cdot r \cdot s$ celdas de memoria (elementos de arreglo) en la cual un elemento particular se determina por tres enteros J, K y L (los subíndices) donde

$$1 \leq J \leq m, \qquad 1 \leq K \leq r, \qquad 1 \leq L \leq s$$

Por ejemplo, supongamos que B es un arreglo tridimensional de $2 \times 4 \times 3$. Entonces, B tiene $2 \cdot 4 \cdot 3 = 24$ elementos, que aparecen en la figura 6-5. Obsérvese que B se dibuja compuesto por tres capas (páginas) de matrices de 2×4. Al primer subíndice lo llamamos la *fila*, el segundo subíndice la *columna* y el tercer subíndice la *página* del elemento; por ejemplo, B(1, 4, 2) es el elemento que está en la primera fila, cuarta columna y segunda página de B. Sin embargo, B está representado internamente como una cadena de 24 celdas de memoria listadas de tal forma, que el primer subíndice varíe primero (más rápidamente), el segundo subíndice varíe posteriormente y así sucesivamente, como en la figura 6-6. Es decir, que los elementos son listados por columnas de la primera página, luego por columnas de la segunda página y luego por columnas de la tercera página. Este orden de almacenamiento se indica mediante flechas en la figura 6-5.

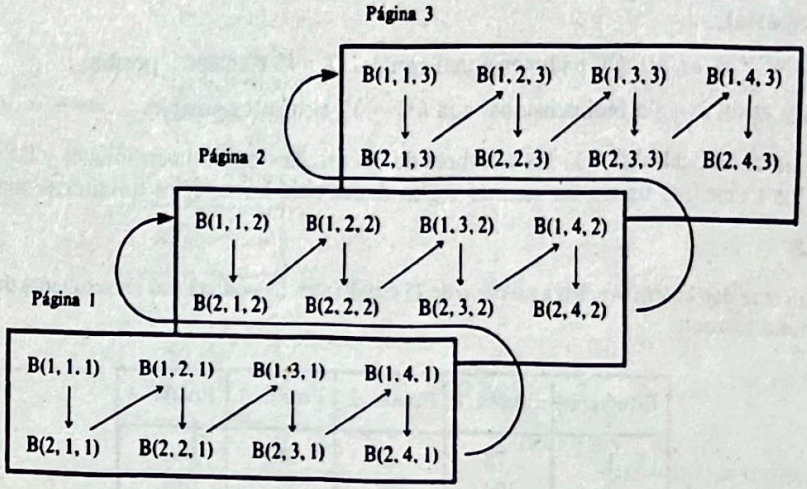


Fig. 6-5

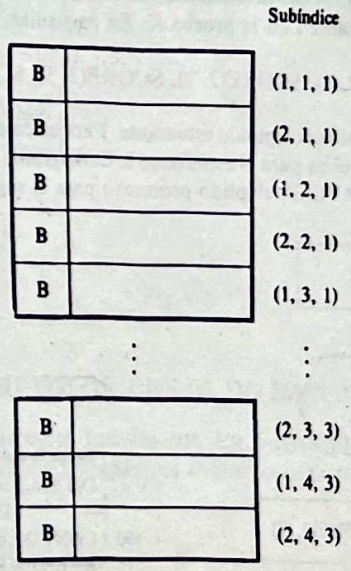


Fig. 6-6

Generalmente, un arreglo p-dimensional de $n_1 \times n_2 \times \ldots \times n_p$ es una lista de $n_1 \cdot n_2 \cdot \ldots \cdot n_p$ celdas de memoria (elementos del arreglo), en la cual un elemento particular se determina por p enteros $K_1, K_2, \ldots, K_p$ (los *subíndices*), donde

$$1 \le K_1 \le n_1,\ 1 \le K_2 \le n_2,\ \ldots,\ 1 \le K_p \le n_p$$

Un elemento particular se designa por el nombre del arreglo seguido por sus p subíndices separados por comas y encerrados entre paréntesis.

Los arreglos multidimensionales pueden declararse en la misma forma que los arreglos unidimensionales. Por ejemplo,

```
DIMENSION ID(25), MONEY(3, 2, 7), TAX(6, 5)
```

informará al compilador que:

1. ID es un arreglo unidimensional con 25 elementos posibles (es decir, el compilador apartará 25 celdas de memoria para ID, aun cuando no necesariamente, todas las celdas sean usadas por el programa);
2. MONEY es un arreglo tridimensional con $3 \cdot 2 \cdot 7 = 42$ elementos posibles;
3. TAX es un arreglo bidimensional con $6 \cdot 5 = 30$ elementos posibles.

Hacemos énfasis en el hecho de que los nombres de los arreglos multidimensionales y las formas de los subíndices de los elementos siguen las mismas reglas dadas para los arreglos unidimensionales.

EJEMPLO 6.3

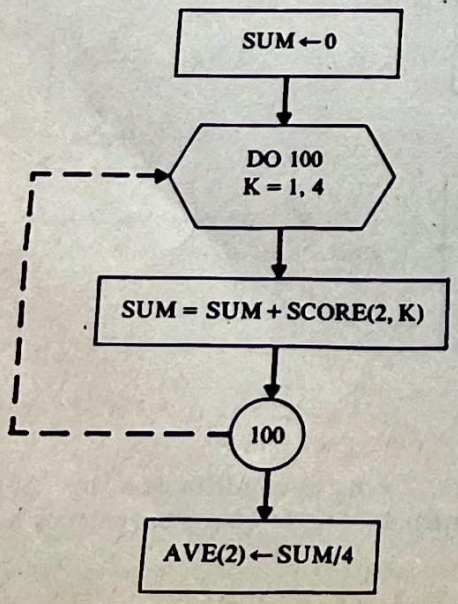
Supongamos ıe se dan cuatro pruebas a un curso de 25 estudiantes. Se podría listar los resultados de las pruebas en una tabla del modo siguiente:

Estudiante	Prueba 1	Prueba 2	Prueba 3	Prueba 4
1	78	83	80	87
2	100	100	92	100
⋮	⋮	⋮	⋮	⋮

Supongamos que esos resultados de las pruebas se almacenan en una matriz real de 25×4, llamada SCORE. Esto es, SCORE (I, K) da el resultado del estudiante I en la prueba K. En particular, la segunda fila

```
SCORE(2, 1), SCORE(2, 2), SCORE(2, 3), SCORE(2, 4)
```

da los resultados de las cuatro pruebas para el segundo estudiante. Para calcular el promedio de un estudiante haremos que AVE(I) denote el promedio de la prueba para el estudiante I. El diagrama de flujo en la figura 6-7 y su equivalente Fortran pueden usarse para encontrar AVE(2), el grado promedio para el segundo estudiante.



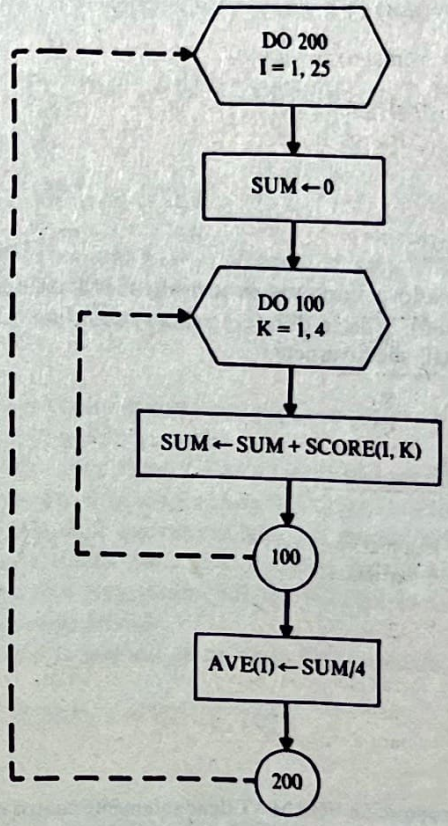

```
      SUM = 0.0
      DO 100 K = 1, 4
            SUM = SUM + SCORE(2, K)
100   CONTINUE
      AVE(2) = SUM/4.0
```

Fig. 6-7

Más generalmente, podemos encontrar AVE(I), el promedio de pruebas para el estudiante I, por el segmento Fortran:

```
      SUM = 0.0
      DO 100 K = 1, 4
            SUM = SUM + SCORE(I, K)
100   CONTINUE
      AVE(I) = SUM/4.0
```

Por consiguiente, para calcular el promedio para cada uno de los 25 estudiantes, necesitaremos un ciclo de DO anidados donde el ciclo DO exterior recorra cada uno de los 25 estudiantes. El diagrama de flujo y su equivalente Fortran aparecen en la figura 6-8.




```
      DO 200 I = 1, 25
            SUM = 0.0
            DO 100 K = 1, 4
                  SUM = SUM + SCORE(I, K)
100         CONTINUE
            AVE(I) = SUM/4.0
200   CONTINUE
```

Fig. 6-8

6.7 ENTRADA/SALIDA DE ARREGLOS, CICLOS DO IMPLICITOS

Supongamos que A y B son arreglos lineales con 4 y 3 elementos respectivamente. Como se anotó previamente, los elementos de A y B pueden aparecer en proposiciones E/S como variables ordinarias. Por ejemplo,

```
READ(5, 10) A(3), B(2), J, X
```

le dice al computador que lea valores para A(3) y B(2), así como para J y X. En particular, se puede entrar los arreglos completos A, B listando todos los elementos individuales; por ejemplo,

```
READ(5, 20) A(1), A(2), A(3), A(4), B(1), B(2), B(3)
```

Evidentemente, esta técnica no sería adecuada si hubiera cientos de elementos en los arreglos. Discutiremos ahora otros métodos para la entrada/salida de arreglos.

(*a*) Nombres de arreglos en proposiciones E/S

Podemos leer o imprimir un arreglo completo usando solamente el nombre del arreglo en una proposición de entrada/salida. Por ejemplo, la anterior proposición READ puede ser reescrita en forma equivalente, como:

```
READ(5, 20) A, B
```

Sin embargo, se deben comprender claramente las acciones que resultan de tal instrucción E/S:

1. El arreglo completo será almacenado o impreso, es decir, cada localización de memoria separada por la proposición, deberá tener un valor asignado. En consecuencia, si SCORE ha sido declarado como un arreglo por

   ```
   DIMENSION SCORE(35)
   ```

   entonces no podremos usar

   ```
       READ(5, 30) SCORE
   30  FORMAT(5F10.1)
   ```

   si deseamos solamente leer 25 valores de SCORE.

2. Los elementos del arreglo tendrán valores asignados o impresos de acuerdo al orden de almacenamiento del arreglo (véanse figuras 6-1, 6-4 y 6-6), es decir, donde el primer subíndice varía más rápidamente, después el segundo subíndice y así, sucesivamente.

EJEMPLO 6.4

Supongamos que C, D y E son arreglos declarados por:

```
DIMENSION C(3), D(2, 3, 2), E(3, 2)
```

Entonces, la pareja WRITE-FORMAT

```
    WRITE(6, 30) C, D, E
30  FORMAT(6X, 4F10.2)
```

instruye al computador para que imprima un total de

$$3 + 2 \cdot 3 \cdot 2 + 3 \cdot 2 = 21$$

valores; 3 para el arreglo C, 12 para D y 6 para E. Puesto que la proposición FORMAT tiene solamente cuatro especificaciones de campo numéricas, se imprimirán cuatro números por línea y la proposición FORMAT se utilizará una y otra vez, hasta que se impriman todos los valores. Luego, la salida aparecerá como en la figura 6-9.

C(1)	C(2)	C(3)	D(1, 1, 1)
D(2, 1, 1)	D(1, 2, 1)	D(2, 2, 1)	D(1, 3, 1)
D(2, 3, 1)	D(1, 1, 2)	D(2, 1, 2)	D(1, 2, 2)
D(2, 2, 2)	D(1, 3, 2)	D(2, 3, 2)	E(1, 1)
E(2, 1)	E(3, 1)	E(1, 2)	E(2, 2)
E(3, 2)			

Fig. 6-9

Observemos que si se usa una proposición FORMAT diferente en el ejemplo anterior, por ejemplo,

```
30  FORMAT(6X, 6F10.2)
```

entonces la salida contendrá seis números por línea. Sin embargo, el orden de los números no se cambiará por la proposición diferente FORMAT.

(*b*) Ciclos DO implícitos

Supóngase que deseamos solamente almacenar o imprimir parte de un arreglo, o que no deseamos almacenar o imprimir un arreglo en su orden de almacenamiento pre-establecido. Posiblemente podríamos hacer esto utilizando un ciclo DO. Por ejemplo, supongamos que AMOUNT es un arreglo lineal con cien elementos posibles. Entonces, el ciclo DO

```
      DO 100 I = 11, N, 1
            READ(5, 10) AMOUNT(I)
100   CONTINUE
```

le dice al computador que lea valores para AMOUNT(11), AMOUNT(12), hasta AMOUNT(N) inclusive, donde N está previamente definido y N ≤ 100. Podemos realizar la misma cosa por medio de la proposición READ

```
READ(5, 20) (AMOUNT(I), I = 11, N, 1)
```

La expresión

```
(AMOUNT(I), I = 11, N, 1)
```

es llamada un *ciclo DO implícito*, puesto que causa que una acción cíclica tenga lugar dentro de la proposición READ. La notación

```
I = 11, N, 1
```

tiene el mismo significado que el que se tiene en el ciclo DO; esto es, 11 es el valor inicial de I, N es el valor de prueba y 1 es el incremento.

Observemos que el ciclo DO implícito es mucho más versátil que el ciclo DO. Específicamente, cada vez que se encuentra la proposición READ en el ciclo DO anterior, otra tarjeta de datos debe de leerse, independientemente de lo que especifique la proposición FORMAT rotulada como 10. En consecuencia, los valores de entrada deben ponerse en tarjetas separadas. Por otra parte, en el ciclo DO implícito, solamente se encuentra una proposición READ; así pues, la entrada puede ponerse en una o más tarjetas, dependiendo de la proposición FORMAT.

La forma general de un ciclo DO implícito de una proposición E/S, es como sigue:

```
(VN(I), I = IN, IE, IC)
```

Donde:

VN	Nombre del arreglo	IE	Valor final o de prueba de I
I	Indice	IC	Incremento
IN	Valor inicial de I		

Hacemos notar que IN, IE e IC pueden ser una constante entera o una variable entera positiva. Entonces, si se usa 1 para IC, ésta puede omitirse. Hacemos énfasis en que el paréntesis exterior y la coma antes del índice son absolutamente necesarios cuando se usa el ciclo DO implícito.

El índice I puede utilizarse para controlar más de un arreglo. Por ejemplo,

```
READ(5, 30) (A(K), B(3, K), C(K, K, 2), K = 2, 9, 3)
```

le dice al computador que recorra la lista de variables, primero con K = 2, luego con K = 5 y posteriormente con K = 8. Entonces, la proposición READ anterior es equivalente a

```
 READ(5, 30) A(2), B(3, 2), C(2, 2, 2), A(5), B(3, 5), C(5, 5, 2),
1              A(8), B(3, 8), C(8, 8, 2)
```

Por lo general, las variables en un ciclo DO implícito no necesitan ser subindicadas, ni involucrar contador.

EJEMPLO 6.5

(*a*) Supongamos que X es un arreglo lineal con cien elementos almacenados en memoria. Podemos imprimir el arreglo con un elemento por línea, por medio de:

```
      DO 100 I = 1, 100
             WRITE(5, 10) X(I)
10           FORMAT(2X, F8.2)
100   CONTINUE
```

ó

```
      WRITE(5, 20) (X(I), I = 1, 100)
20    FORMAT(2X, F8.2)
```

Podemos imprimir el arreglo, con cuatro elementos por línea, así:

```
      DO 100 I = 1, 100, 4
             WRITE(5, 10) X(I), X(I + 1), X(I + 2), X(I + 3)
10           FORMAT(4(2X, F8.2))
100   CONTINUE
```

ó,

```
      WRITE(5, 20) (X(I), I = 1, 100)
20    FORMAT(4(2X, F8.2))
```

Vemos, entonces, cómo el ciclo DO implícito aventaja en versatilidad al ciclo DO para operaciones E/S.

(*b*) Consideremos la pareja WRITE-FORMAT:

```
      WRITE(6, 40) (A(I), B, X(3), Y(I, I), I = 1, 8, 3)
40    FORMAT(6(2X, F8.2))
```

El ciclo DO dentro de la proposición WRITE se ejecuta 3 veces: primero con I = 1, luego I = 4 y finalmente con I = 7. Entonces, la proposición WRITE es equivalente a:

```
      WRITE(6, 40) A(1), B, X(3), Y(1, 1), A(4), B, X(3),
     1             Y(4, 4), A(7), B, X(3), Y(7, 7)
```

Obsérvese que B y X(3) aparecen tres veces. La proposición FORMAT hará que se impriman seis números por línea, luego la salida aparecerá como sigue:

```
A(1)    B       X(3)    Y(1, 1)    A(4)    B
X(3)    Y(4, 4) A(7)    B          X(3)    Y(7, 7)
```

(*c*) El segmento de programa

```
      WRITE(6, 50) (I, X(I), I = 1, 3)
50    FORMAT(1X, 'A(', I1, ') = ', F10.2)
```

imprimirá

```
A(1) = XXXXXXX.XX
A(2) = XXXXXXX.XX
A(3) = XXXXXXX.XX
```

(*c*) Ciclos DO implícitos anidados

El ciclo DO implícito descrito anteriormente, puede ser generalizado para controlar arreglos multidimensionales. (Esta generalización es análoga a los ciclos DO anidados.) Supongamos, por ejemplo, que F ha sido declarado como un arreglo de 3 × 2 y que el programa contiene la proposición

```
READ(5, 10) ((F(I, J), J = 1, 2), I = 1, 3)
```

El índice exterior I es análogo al índice de un DO exterior y el índice interior J es análogo al índice de un ciclo DO interior. Esto es, el ciclo DO implícito interior se ejecuta para cada valor del índice del ciclo DO implícito exterior. Luego, la proposición READ anterior es equivalente a

```
READ(5, 10) (F(1, J), J = 1, 2), (F(2, J), J = 1, 2), (F(3, J), J = 1, 2)
```

la cual es equivalente a

```
READ(5, 10) F(1, 1), F(1, 2), F(2, 1), F(2, 2), F(3, 1), F(3, 2)
```

Obsérvese que la anterior proposición READ lee valores de F fila por fila. Por otra parte, la proposición

```
READ(5, 20) ((F(I, J), I = 1, 3), J = 1, 2)
```

(donde los índices I y J se han intercambiado) es equivalente a

```
READ(5, 20) F(1, 1), F(2, 1), F(3, 1), F(1, 2), F(2, 2), F(3, 2)
```

y, en este caso, los valores de F se han leído por columnas.

Los ciclos DO implícitos anidados pueden tener combinaciones más complejas y también pueden aparecer en proposiciones WRITE.

EJEMPLO 6.6

(*a*) Considere la proposición

```
WRITE(6, 11) ((B(L, M), L = 2, 10, 2), M = 1, 11, 3)
```

Primero, el ciclo DO interior (B(L, M), L = 2, 10, 2) se ejecuta con M = 1, luego con M = 4, luego con M = 7 y finalmente M = 10. Por lo tanto, la proposición es equivalente a

```
      WRITE(6, 11) (B(L, 1), L = 2, 10, 2), (B(L, 4), L = 2, 10, 2),
     1             (B(L, 7), L = 2, 10, 2), (B(L, 10), L = 2, 10, 2)
```

lo cual es equivalente a

```
      WRITE(6, 11) B(2, 1), B(4, 1), B(6, 1), B(8, 1), B(10, 1), B(2, 4), B(4, 4),
     1             B(6, 4), B(8, 4), B(10, 4), B(2, 7), B(4, 7), B(6, 7), B(8, 7),
     1             B(10, 7), B(2, 10), B(4, 10), B(6, 10), B(8, 10), B(10, 10)
```

Obsérvese que en el ejemplo precedente, el segundo subíndice M en B(L, M) aparece en el ciclo DO exterior, mientras que el primer subíndice I en F(I, J) aparece en el ciclo DO exterior.

(*b*) Suponga que B es un arreglo de 4 × 5 × 3 y suponga que deseamos imprimir los valores de B por filas de la primera página, luego las filas de la segunda página y finalmente las filas de la tercera página. Esto se puede hacer con la pareja WRITE-FORMAT:

```
      WRITE(6, 50) (((B(K1, K2, K3), K2 = 1, 5), K1 = 1, 4), K3 = 1, 3)
 50   FORMAT(5(2X, F8.2))
```

En efecto, la proposición FORMAT permitirá imprimir solamente cinco números por línea; de modo que las diferentes filas de B aparecerán en diferentes líneas de las páginas de impresión.

6.8 EJEMPLO DE TECNICA DE PROGRAMACION

Dado cualquier problema, puede haber diferentes modos (algoritmos) para formularlo y resolverlo. Algunos son mejores que otros; es decir, algunos pueden requerir menos memoria y/o menos operaciones. El capítulo 8 describe varias técnicas de programación. Aquí consideraremos un ejemplo sencillo que ilustrará las sutilezas que pueden tenerse en cuenta para alcanzar su solución.

Supongamos que deseamos mover los elementos A(1), A(2),..., A(N) corriendo la lista en una posición:

de tal modo, que A(2) contenga el antiguo A(1), A(3) contenga el antiguo A(2) y así, sucesivamente.

Lo que intuitivamente se puede hacer es asignar primero A(1) a A(2), luego A(2) a A(3) y así sucesivamente, hasta que asignemos A(N) a A(N + 1). Esto podría hacerse mediante el ciclo DO

```
      DO 100 I = 1, N
             A(I + 1) = A(I)
100   CONTINUE
```

Sin embargo, este procedimiento *no funcionará*. La razón es que la primera asignación de A(1) a A(2) borra el valor original de A(2), y entonces, la asignación siguiente de A(2) a A(3) almacenará el valor original de A(1) en A(3). Por lo tanto, el procedimiento anterior hará que el valor de A(1) sea asignado a todas las localizaciones A(2), A(3),..., A(N + 1).

Una forma de corregir el procedimiento anterior es guardar temporalmente los valores originales de A en otras localizaciones. Esto se puede hacer usando otro arreglo, digamos B. Específicamente, utilizamos un ciclo DO para asignar A(I) a B(I + 1), y luego otro ciclo DO para transferir nuevamente los elementos a A, asignando B(I + 1) a A(I + 1). Sin embargo, esto significa que debemos usar el doble de las celdas de memoria de A para mover los elementos de A en una posición; es decir, si A contiene 1000 celdas de memoria, necesitaremos otras 1000 celdas de memoria para B.

Otra forma requiere solamente dos celdas de memoria adicionales, de la siguiente manera: primero, guardamos A(2) en una localización temporal, TEMPA, antes de asignar A(1) a A(2), y luego guardamos A(3) en una localización de almacenamiento temporal TEMPB antes de asignar a TEMPA a A(3):

```
TEMPA ← A(2)
A(2)  ← A(1)
TEMPB ← A(3)
A(3)  ← TEMPA
```

Obsérvese que ahora A(3) contiene el valor original de A(2). Usaremos TEMPA de nuevo para guardar A(4) antes de asignar TEMPB a A(4), y así sucesivamente. Lo anterior, puede realizarse mediante un ciclo DO. Aunque se usan solamente dos celdas de memoria adicionales, el algoritmo requiere 2N – 1 asignaciones.

Sin embargo, hay otro camino mejor que no requiere memoria extra y solamente necesita N asignaciones., simplemente movemos A(N) a A(N + 1), luego A(N – 1) a A(N) y así, sucesivamente. Esto es, damos la siguiente secuencia de instrucciones

```
A(N + 1) ← A(N)
A(N)     ← A(N – 1)
   ⋮
A(2)     ← A(1)
```

El equivalente Fortran se da seguidamente:

```
      DO 100 J = 1, N
            K = N + 1 – J
            A(K + 1) = A(K)
100   CONTINUE
```

ó simplemente

```
      DO 100 J = 1, N
            A(N – J + 2) = A(N – J + 1)
100   CONTINUE
```

(Ver ejemplo 6.2, para ver por qué se introduce la nueva variable K.)

Esto requiere el uso del elemento A(N+1), lo que no es correcto.

Problemas resueltos

ARREGLOS

6.1 Determinar el número de elementos de los arreglos A, B y K de acuerdo a la proposición

```
DIMENSION A(25), B(3, 8, 4), K(7, 5)
```

A tiene 25 elementos.
B tiene 3.8.4 = 96 elementos.
K tiene 7.5 = 35 elementos.

6.2 Encontrar el número de elementos de un arreglo definido por:

```
DIMENSION X(2, 4, 3)
```

Describir la salida si se ejecuta el siguiente par de proposiciones:

```
      WRITE(6, 10) X
10    FORMAT(6(2X, F12.2))
```

X tiene 2.4.3 = 24 elementos. Estos se imprimirán en su orden de almacenamiento, es decir, de tal forma que el primer subíndice (fila) varía más rápidamente, luego el siguiente subíndice (columna) y por último el tercer subíndice (página). Además, habrá seis números por línea. Luego la salida aparecerá como sigue:

X(1, 1, 1)	X(2, 1, 1)	X(1, 2, 1)	X(2, 2, 1)	X(1, 3, 1)	X(2, 3, 1)
X(1, 4, 1)	X(2, 4, 1)	X(1, 1, 2)	X(2, 1, 2)	X(1, 2, 2)	X(2, 2, 2)
X(1, 3, 2)	X(2, 3, 2)	X(1, 4, 2)	X(2, 4, 2)	X(1, 1, 3)	X(2, 1, 3)
X(1, 2, 3)	X(2, 2, 3)	X(1, 3, 3)	X(2, 3, 3)	X(1, 4, 3)	X(2, 4, 3)

6.3 Supongase que N y un arreglo lineal A con N elementos están en memoria. Escriba un segmento de programa Fortran que:

(*a*) Calcule la suma de los cuadrados:

$$SUM = A_1^2 + A_2^2 + \cdots + A_n^2$$

(*b*) Calcule el producto

$$PROD = (1 - A_1)(1 - A_2) \ldots (1 - A_N)$$

(*a*) Primero hacemos a SUM = 0.0 antes de realizar la sumatoria:

```
      SUM = 0.0
      DO 100 K = 1, N
            SUM = SUM + A(K)**2
100   CONTINUE
```

(*b*) Primero hacemos a PROD = 1.0 antes de realizar la multiplicación. (Esto es análogo a hacer SUM = 0.0 en (*a*).)

```
      PROD = 1.0
      DO 100 K = 1, N
            PROD = PROD*(1.0 - A(K))
100   CONTINUE
```

6.4 Encontrar la salida en el siguiente programa:

```
      INTEGER A(10)
      DO 100 I = 1, 10, 2
            A(I) = 2*I - 3
            A(I + 1) = I**2 - 5
100   CONTINUE
      DO 200 I = 2, 10, 3
            A(I) = A(I + 2) - A(I)
200   CONTINUE
      WRITE(6, 10) A
 10   FORMAT(1X, 5I8)
      STOP
      END
```

La primera proposición declara a A como un arreglo lineal entero con 10 elementos. Se ejecuta el primer ciclo DO:

(*i*) Primero, para I = 1. Esto conduce a A(I) = −1 y A(2) = −4.
(*ii*) Luego, para I = 3. Esto conduce a A(3) = 3 y A(4) = 4.
(*iii*) Luego, para I = 5. Esto conduce a A(5) = 7 y A(6) = 20.
(*iv*) Luego, para I = 7. Esto conduce a A(7) = 11 y A(8) = 44.
(*v*) Finalmente, para I = 9. Esto conduce a A(9) = 15 y A(10)=76.

Se ejecuta el segundo ciclo DO:

(*i*) Primero, para I = 2. Esto cambia el valor de A(2), como sigue:

$$A(2) \leftarrow A(4) - A(2) = 4 - (-4) = 8$$

(*ii*) Luego, para I = 5. Esto cambia el valor de A(5), como sigue:

$$A(5) \leftarrow A(7) - A(5) = 11 - 7 = 4$$

(*iii*) Finalmente, para I = 8. Esto cambia el valor de A(8), como sigue:

$$A(8) \leftarrow A(10) - A(8) = 76 - 44 = 32$$

El par de proposiciones WRITE-FORMAT imprime los valores de A con cinco enteros sobre una línea en campos con longitud 8. Luego, la salida aparecerá como sigue:

```
  -1       8       3       4       4
  20      11      32      15      76
```

6.5 Supongamos que N y un arreglo lineal A con N elementos están en localizaciones de memoria. Escribir un segmento de programa Fortran que:

(*a*) Intercambie los valores de A_1 y A_2, intercambie los valores de A_3 y A_4 y así sucesivamente. (Aquí convenimos que N es par.)
(*b*) Cíclicamente permute el valor de A, de tal forma que A_1 contenga el valor original de A_2, A_2 contenga el valor original de A_3 y así sucesivamente, con A_N que contiene el valor original de A_1.

(*a*) Recalquemos primero que intercambiamos los valores de las variables X e Y, utilizando una localización de almacenamiento temporal T, así:

```
T = X
X = Y
Y = T
```

Entonces, tenemos el segmento de programa:

```
      DO 100 K = 1, N, 2
            T = A(K)
            A(K) = A(K + 1)
            A(K + 1) = T
100   CONTINUE
```

(*b*) Primero, almacenamos temporalmente A_1 en T; luego, hacemos $A_1 = A_2, A_2 = A_3, \ldots A_{N-1} = A_N$; finalmente, A_N = T. Entonces, tenemos:

```
      T = A(1)
      NN = N - 1
      DO 100 K = 1, NN
            A(K) = A(K + 1)
100   CONTINUE
      A(N) = T
```

Obsérvese de nuevo que usamos la proposición NN = N−1 como parámetro del DO.

PROGRAMAS

6.6 Considerar un grupo de números $x_1, x_2, \ldots, x_n$. La media aritmética $\bar{x}$, la varianza y la desviación típica σ están definidas por:

$$\bar{x} = (x_1 + x_2 + \cdots + x_n)/n = \left(\sum_{k=1}^{n} x_k\right)\Big/ n$$

$$\text{varianza} = \frac{\sum_{k=1}^{n} (x_k - \bar{x})^2}{n}$$

$$\sigma = \sqrt{\text{varianza}}$$

Supongamos que N números están almacenados en un arreglo lineal A. Escribir un segmento de programa Fortran con el cual almacenaremos la media, la varianza y la desviación típica de los números en AVE, VAR y SD, respectivamente. (Compárese con el problema 5.9 donde se utilizó la segunda fórmula para la varianza.)

El segmento de programa es como sigue:

```
      SUM1 = 0.0
      DO 100 K = 1, N
            SUM1 = SUM1 + A(K)
100   CONTINUE
      AVE = SUM1/FLOAT(N)
      SUM2 = 0.0
      DO 200 K = 1, N
            SUM2 = SUM2 + (A(K) - AVE)**2
200   CONTINUE
      VAR = SUM2/FLOAT(N)
      SD = SQRT(VAR)
```

6.7 Una cadena de almacenes de departamentos tiene seis almacenes y cada almacén tiene diez departamentos iguales. Las ventas semanales de la cadena están almacenadas en un arreglo de 6×10, llamado SALES. [Aquí, SALES(I, J) denota las ventas semanales en el almacén I del departamento J.] Escribir un segmento de programa Fortran que:

(*a*) Imprima las ventas semanales totales de cada almacén.
(*b*) Imprima las ventas totales semanales de cada departamento.
(*c*) Imprima las ventas totales semanales de la cadena de almacenes.

Convenimos que STORE y DEPT han sido declarados arreglos lineales mediante

```
DIMENSION STORE(6), DEPT(10)
```

(*a*) Sumamos las ventas para un I dado, después de que inicialmente hayamos establecido que STORE(I) = 0.

```
      WRITE(6, 10)
10    FORMAT('1', 5X, 'ALMACEN', 5X, 'VENTAS'//)
      DO 99 I = 1, 6
            STORE(I) = 0.0
            DO 88 J = 1, 10
                  STORE(I) = STORE(I) + SALES(I, J)
88          CONTINUE
            WRITE(6, 20) I, STORE(I)
20          FORMAT(6X, I3, 5X, F9.2)
99    CONTINUE
```

(*b*) Sumamos las ventas para un J dado, después de que inicialmente hayamos establecido que DEPT(J) = 0.

```
      WRITE(6, 30)
30    FORMAT('0', 2X, 'DEPTO', 3X, 'VENTAS'//)
      DO 77 J = 1, 10
            DEPT(J) = 0.0
            DO 66 I = 1, 6
                  DEPT(J) = DEPT(J) + SALES(I, J)
66          CONTINUE
            WRITE(6, 40) J, DEPT(J)
40          FORMAT(3X, I5, 5X, F9.2)
77    CONTINUE
```

(*c*) Podemos sumar los valores almacenados en el arreglo SALES, o en el arreglo STORE o en el arreglo DEPT (o en cada uno de los tres para un chequeo de los resultados.) Nosotros lo hacemos para el arreglo STORE.

```
      CHAIN = 0.0
      DO 55 K = 1, 6
            CHAIN = CHAIN + STORE(K)
55    CONTINUE
      WRITE(6, 50) CHAIN
50    FORMAT('0', 'TOTAL VENTAS SEMANALES'/6X, F9.2)
```

6.8 Imprimir los 50 primeros términos de la serie de Fibonacci, de tal forma que haya cinco números por línea.

Recordemos que la secuencia de Fibonacci es

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

esto es, los dos primeros términos son 1, y cualquier otro término se obtiene sumando los dos términos anteriores:

$$2 = 1 + 1, \quad 3 = 1 + 2, \quad 5 = 2 + 3, \quad 8 = 3 + 5, \ldots$$

Luego, dejamos

$$J(1) = 1, \quad J(2) = 1, \quad \text{y} \quad J(I) = J(I-2) + J(I-1)$$

para $I > 2$. El programa es como sigue:

```
      DIMENSION J(50)
      J(1) = 1
      J(2) = 1
      DO 100 I = 3, 50
            J(I) = J(I - 2) + J(I - 1)
100   CONTINUE
      WRITE(6, 10)
10    FORMAT('1', 15X, 'SECUENCIA DE FIBONACCI'//)
      WRITE(6, 20) J
20    FORMAT(1X, 5(3X, I6))
      STOP
      END
```

Obsérvese que hay solamente cinco especificaciones numéricas en la proposición FORMAT numerada con 20, así que solamente se imprimen cinco números en cada línea.

6.9 (*a*) Supongamos que B es un arreglo lineal con M elementos. Escribir un segmento de programa que encuentre la localización J, tal que B(J) contenga el mayor valor absoluto de B. (Véase ejemplo 6.1.)

(*b*) Supongamos que A es una matriz de M × N localizada en memoria. Escribir un segmento de programa Fortran que:

(*i*) encuentre la posición J, tal que J contenga el mayor valor absoluto de la primera columna, y que

(*ii*) intercambie la primera fila con la fila J.

(Entonces, A(1, 1) contendrá el mayor valor absoluto en la primera columna.)

(*a*) Establecemos J = 1 y BIG = ABS(B(K)) y luego comparamos BIG con los otros elementos de B. Si BIG es menor que ABS(B(K)), entonces tendremos BIG = ABS(B(K)) y cambiamos J por K. El programa es como sigue:

```
      J = 1
      BIG = ABS(B(1))
      DO 100 K = 1, M
            IF(BIG.GE.ABS(B(K))) go to 100
            BIG = ABS(B(K))
            J = K
  100 CONTINUE
```

El índice K se hace inicialmente igual a 1 para tener en cuenta la posibilidad de que M pueda ser 1.

(*b*) El segmento de programa tiene dos partes: (*i*) Primero, encuentre el valor de J. (*ii*) Si J ≠ 1, entonces intercambie la primera fila con la fila J.

```
C
C       ENCONTRAR LA POSICION J CON EL MAYOR VALOR ABSOLUTO
C
        J = 1
        BIG = ABS(A(1,1))
        DO 100 K = 1, M
              IF(BIG.GE.ABS(A(K,1)))GO TO 100
              BIG = ABS(A(K,1))
              J = K
  100   CONTINUE
        IF(J.EQ.1) GO TO 500
C
C       INTERCAMBIAR LA FILA 1 Y LA FILA J
C
        DO 200 L = 1, N
              TEMP = A(1, L)
              A(1, L) = A(J, L)
              A(J, L) = TEMP
  200   CONTINUE
  500   ***********************************************
```

Obsérvese la similitud entre la primera parte del segmento de programa y la parte (*a*).

6.10 Escribir un segmento de programa que inserte un elemento D en la posición K del arreglo A(1), A(2),..., A(N).

Antes de asignar D a A(K), debemos mover el segmento del arreglo A(K), A(K+1),..., A(N) una posición hacia abajo. Esto se hace empezando por el final del arreglo, es decir, primero asignamos A(N) a A(N+1), luego A(N − 1) a A(N), y así sucesivamente, hasta que A(K) se asigne a A(K + 1) (véase sección 6.8.) Obsérvese primero que deben moverse N − K + 1 elementos y no N − K, puesto que también estamos moviendo a A(K). Usemos un ciclo DO con índice J = 1, 2,..., N − K + 1. La relación entre el índice J y la acción es como sigue:

Valores de J	Acción
1	A(N + 1) ← A(N)
2	A(N) ← A(N − 1)
3	A(N − 1) ← A(N − 2)
...	.....
N − K + 1	A(K + 1) ← A(K)

El segmento de programa es como sigue:

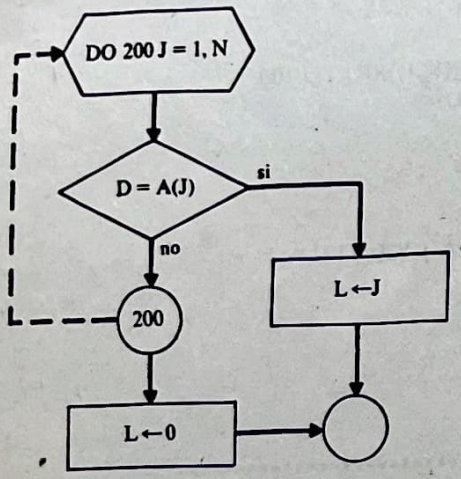
```
      NN = N - K + 1
      DO 100 J = 1, N, N
            L = N - J + 1
            A(L + 1) = A(L)
100   CONTINUE
      A(K) = D
```

ó simplemente

```
      NN = N - K + 1
      DO 100 J = 1, NN
            A(N - J + 2) = A(N - J + 1)
100   CONTINUE
      A(K) = D
```

6.11 (*Búsqueda lineal*) Dado un arreglo A(1), A(2),..., A(N), y un elemento D, escribir un segmento de programa que encuentre la localización L de D cuando D esté en el arreglo o haga L = 0 cuando D no esté en el arreglo.

Como no hay otra información acerca del arreglo A, la forma más intuitiva para resolver el problema es comparar D con cada uno de los elementos de A; esto es, probamos si D = A(1), luego probamos si D = A(2) y así, sucesivamente. Este método se llama *búsqueda lineal* o *búsqueda secuencial*, puesto que investiga a D probándolo uno a uno con los elementos de A. (Otros métodos de búsqueda son examinados en el capítulo 8.) Claramente, requeriremos a lo sumo N comparaciones para localizar a D, y siempre tendremos N comparaciones cuando D no está en el arreglo. La figura 6-10 muestra el diagrama de flujo para el algoritmo y su traducción Fortran.




```
      DO 200 J = 1, N
            IF(D.EQ.A(J)) GO TO 300
200   CONTINUE
      L = 0
      GO TO 400
300   L = J
400
      .
      .
      .
```

Fig. 6-10

6.12 (*Ordenamiento*) Se dice que un arreglo es *ordenado*, si sus elementos están arreglados en algún orden. Escribir un segmento de programa que ordene el arreglo A(1), A(2),..., A(N), de tal forma que sus elementos estén en orden creciente.

Arreglamos A mediante el algoritmo conocido como "Ordenamiento por burbuja". (Otros métodos de ordenamiento se discuten en el capítulo 8.) Comparamos primero a(1) con A(2) y los arreglamos en el orden deseado, es decir, de tal forma que A(1) < A(2). Luego, comparamos A(2) y A(3) y los arreglamos de tal forma que A(2) < A(3). Luego, comparamos A(3) y A(4) y los arreglamos de tal forma que A(3) < A(4). Esto se continúa hasta que A(N - 1) < A(N). Cuando "hayamos recorrido" los elementos, el mayor elemento ha sido "empujado" a la posición N.

El proceso anterior se repite para los elementos A(1), A(2),..., A(N - 1). Después de recorrer estos elementos, el segundo elemento mayor es empujado a la posición N-1, y así sucesivamente. Después de N - 1 recorridos, el arreglo A estará ordenado en orden creciente.

Establezcamos formalmente el algoritmo:

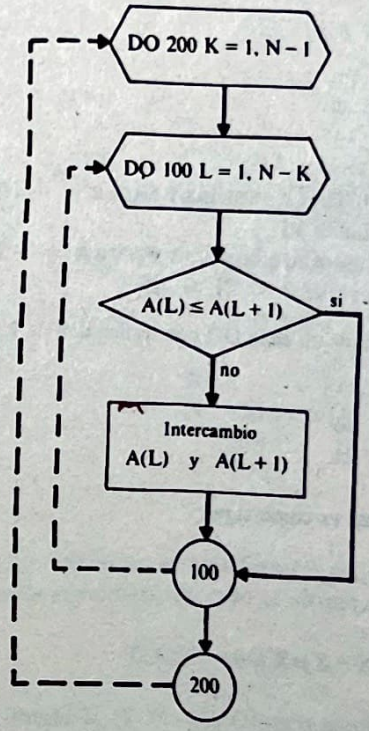
Paso 1. Para L = 1, 2,..., N – 1, comparar A(L) con A(L + 1) y ordenarlos de tal forma que A(L) < A(L + 1).

Paso 2. Para L = 1, 2,..., N – 2, comparar A(L) con A(L + 1) y ordenarlos de tal forma que A(L) < A(L + 1).

⋮ .....

Paso N – 1. Para L = 1, comparar A(L) con A(L + 1) y ordenarlos de tal forma que A(L) < A(L + 1).

Obsérvese que la acción en cada paso es idéntica. Por consiguiente, podemos tener un ciclo DO controlado por un índice K con K = 1, 2,..., N – 1 (correspondiente a los N – 1 pasos), y dentro de este ciclo DO, un ciclo DO interior con índice L = 1, 2,...,N – K. La figura 6-11 muestra el diagrama de flujo para el algoritmo y su traducción Fortran.




```
C
C       ORDENAMIENTO POR BURBUJA
C
        NN = N - 1
        DO 200 K = 1, NN
              JJ = N - K
              DO 100 L = 1, JJ
                    IF(A(L).LE.A(L + 1)) GO TO 100
                    TEMP = A(L)
                    A(L) = A(L + 1)
                    A(L + 1) = TEMP
100           CONTINUE
200     CONTINUE
```

Fig. 6-11

6.13 Escribir un programa que lea 25 enteros positivos diferentes, $N_1, N_2, \ldots, N_{25}$, e imprimir en líneas separadas, cada par de enteros cuya suma sea 75. (Nota: Si 25 y 50 son dos de esos enteros, entonces se quiere imprimir 25 y 50 ó 50 y 25, pero no ambas formas.)

Primero, comparamos N_1 con $N_2, N_3, \ldots, N_{25}$ para ver si

$$N_1 + N_K = 75$$

donde K > 1. Luego, comparamos N_2 con $N_3, N_4, \ldots, N_{25}$ para ver si

$$N_2 + N_K = 75$$

donde K > 2 y así, sucesivamente. En otras palabras, para J = 1, 2,..., 24, verificamos si

$$N_J + N_K = 75$$

para K > J. Esto puede hacerse con ciclos DO anidados, como sigue:

```
      DIMENSION N(25)
      READ(5, 10) N
   10 FORMAT(5I5)
      DO 100 J = 1, 24
            JJ = J + 1
            DO 200 K = JJ, 25
                  IF(N(J) + N(K).NE.75) GO TO 200
                  WRITE(6, 20) N(J), N(K)
   20             FORMAT(1X, I5, 3X, I5)
  200       CONTINUE
  100 CONTINUE
      STOP
      END
```

6.14 Considerar la ecuación polinomial

$$y = 2x^4 - 5x^3 + 6x^2 - 8x + 9$$

(*a*) Escribir un segmento de programa Fortran que almacene e imprima y para $x = -10, -9, \ldots, 10$, esto es, para valores enteros de x desde -10 hasta 10.

(*b*) Escribir un segmento de programa Fortran que almacene e imprima y para $x = -N, -N+1, \ldots, N$, donde N es leído como un dato. (Asumir que $0 < N \leq 25$.)

(*a*) Nótese que hay 21 valores de X. Por lo tanto, emplearemos un ciclo DO con índice K = 1, 2,..., 21. La relación entre X y K es como sigue:

Valor de X:	−10,	−9,	−8,	...,	9,	10
Valor de K:	1,	2,	3,	...,	20,	21

Obsérvese que X = − 11 + K. El segmento de programa es como sigue:

```
      DIMENSION Y(21)
      DO 100 K = 1, 21
            X = -11 + K
            Y(K) = (((2.*X - 5.)*X + 6.)*X - 8.)*X + 9.
            WRITE(6, 10) X, Y(K)
   10       FORMAT(1X, F5.1, 2X, F8.1)
  100 CONTINUE
```

(*b*) Ahora hay 2N + 1 valores de X. (Nótese que $2N + 1 \leq 51$, puesto que $N \leq 25$.) Por lo tanto, emplearemos un ciclo DO con índice K = 1, 2,..., 2N+1. Aquí, X y K están relacionados por X = − N—1 + K.

```
      DIMENSION Y(51)
      READ(5, 20) N
   20 FORMAT(I5)
      NN = 2*N + 1
      DO 88 K = 1, NN
            X = -N - 1 + K
            Y(K) = (((2.*X - 5.)*X + 6.)*X - 8.)*X + 9.
            WRITE(6, 30) X, Y(K)
   30       FORMAT(1X, F5.1, 2X, F8.1)
   88 CONTINUE
```

Obsérvese que usamos la fórmula de Horner para calcular Y. (Ver sección 8.6.)

6.15 (*Frecuencia de distribución*) Un paquete con 200 tarjetas contiene un entero positivo K por tarjeta, donde $K \leq 25$. Escribir un programa que imprima los enteros positivos de 1 a 25 y el número de veces que cada uno de los enteros está en el paquete.

El algoritmo que usamos es como sigue: declaramos a N como un arreglo lineal con 25 elementos y consideramos que N(K) es el número de veces que ha aparecido el valor K. Inicialmente, tenemos cada N(K) = 0 y cuando aparezca el valor K, aumentamos a N(K) en uno. Por ejemplo, inicialmente, tenemos N(8) = 0, y si K = 8 ocurre 14 veces, entonces N(8) = 14 después de que todas las tarjetas hayan sido leídas. El programa es como sigue:

```
      DIMENSION N(25)
C
C     HACER CADA N(K)  0
C
      DO 100 K = 1, 25
            N(K) = 0
  100 CONTINUE
C
C     LECTURA Y CONTEO DE LOS NUMEROS
C
      DO 200 J = 1, 200
            READ(5, 10) K
   10       FORMAT(I5)
            N(K) = N(K) + 1
  200 CONTINUE
C
C     IMPRIMIR LA DISTRIBUCION
C
      DO 300 K = 1, 25
            WRITE(6, 20) K, N(K)
   20       FORMAT(6X, 2(I4, 5X))
  300 CONTINUE
      STOP
      END
```

6.16 Considerar un arreglo lineal A con N elementos. La primera diferencia DA de A se obtiene restando cada elemento, excepto el último, del siguiente elemento del arreglo:

$$DA(K) = A(K + 1) - A(K)$$

(donde $K \leq N - 1$). Obsérvese que DA tiene N – 1 elementos. La segunda diferencia D2A de A se define como la primera diferencia de DA, y así, sucesivamente.

Supongamos que A está en la memoria. Escriba un segmento de programa que encuentre la primera diferencia DA, la segunda D2A y la tercera diferencia D3A de la secuencia A:

2, 8, −3, 5, 9, −4, 8, 0, −8, 16

y luego imprima A, DA, D2A y D3A.

Las diferencias se pueden mostrar como siguen:

```
 2     8    −3     5     9    −4     8     0    −8    16
    6   −11     8     4   −13    12    −8    −8    24
  −17    19    −4   −17    25   −20     0    32
     36   −23   −13    42   −45    20    32
```

Esto es, la segunda fila de números es la primera diferencia, la tercera fila es la segunda diferencia y la última fila es la tercera diferencia.

El programa es como sigue:

```
C
C       ENCONTRAR LA PRIMERA DIFERENCIA
C
        N = N - 1
        DO 100 K = 1, N
              DA(K) = A(K + 1) - A(K)
  100   CONTINUE
C
C       ENCONTRAR LA SEGUNDA DIFERENCIA
C
        N = N - 1
        DO 200 K = 1, N
              D2A(K) = DA(K + 1) - DA(K)
  200   CONTINUE
C
C       ENCONTRAR LA TERCERA DIFERENCIA
C
        N = N - 1
        DO 300 K = 1, N
              D3A(K) = D2A(K + 1) - D2A(K)
  300   CONTINUE
C
C       IMPRIMIR LOS ARREGLOS
C
        WRITE(6, 10) A
        WRITE(6, 10) DA
        WRITE(6, 10) D2A
        WRITE(6, 10) D3A
   10   FORMAT(1X, 10(2X, F8.2))
```

Si los arreglos DA, D2A, D3A no son necesarios más tarde en el programa, podríamos haber utilizado ciclos DO anidados.

CICLOS DO IMPLICITOS

6.17 Reescribir cada proposición con el ciclo DO implícito, de tal forma que no contenga errores.

(*a*) READ(5, 10), (A(K) K = 1, 20)
(*b*) WRITE(6, 20), A(I, I), S, B(I), I = 1, N, 3
(*c*) READ(5, 30) (A(J), J = 1, 7), B, (C(K), K = 1, M, L)
(*d*) WRITE(6, 40) (A(L, K + 3), K = 1, 7), L = 1, 5)

(*a*) READ(5, 10) (A(K), K = 1, 20)
(*b*) WRITE(6, 20) ((A(I, I), S, B(I)), I = 1, N, 3)
(*c*) No hay errores.
(*d*) No podemos usar a K+3 como parámetro; en cambio podemos escribir:

```
KK = K + 3
WRITE(6, 40) ((A(L, KK), KK = 4, 10), L = 1, 5)
```

6.18 Describir la salida si se ejecuta cada pareja WRITE-FORMAT:

(*a*)
```
      WRITE(6, 10) (A(K), K = 2, 25, 3)
10    FORMAT(1X, 5(3X, F10.2))
```

(*b*)
```
      WRITE(6, 10) (A(L), B, C(3, L), L = 1, 8, 2)
10    FORMAT(1X, 5(3X, F10.2))
```

(*c*)
```
      WRITE(6, 10) (B(J), J = 1, 4), (B(K), K = 3, 10, 4)
10    FORMAT(1X, 5(3X, F10.2))
```

(*a*) Nótese que 2 es el valor inicial de K, 3 el incremento y 25 el valor límite. Por consiguiente, A(2), A(5), A(8), A(11), A(14), A(17), A(20) y A(23) se imprimen a cinco números por línea. Luego, la salida aparece como:

```
A(2)    A(5)    A(8)    A(11)    A(14)
A(17)   A(20)   A(23)
```

(*b*) Nótese que el computador ejecuta el ciclo DO implícito, primero con L=1, luego con L=3, luego con L=5, y finalmente con L=7. Entonces, se imprimen, A(1), B, C(3, 1), A(3), B, C(3, 3), A(5), B, C(3, 5), (A(7), B y C(3, 7) de a cinco números por cada línea:

```
A(1)    B       C(3, 1)   A(3)     B
C(3, 3) A(5)    B         C(3, 5)  A(7)
B       C(3, 7)
```

(*c*)
```
B(1)    B(2)    B(3)    B(4)    B(3)
B(7)
```

6.19 Describir la salida si se ejecutan las siguientes proposiciones:

```
      WRITE(6, 20) ((A(J, K), J = 2, 8, 3), K = 1, 4)
20    FORMAT(IX, 5(3X, F10.2)
```

El DO implícito interior da: A(2, K), A(5, K), A(8, K). Luego, para K=1, 2, 3, 4, habrá un total de 12 valores que serán impresos, cinco por línea. Entonces, la salida aparece como sigue:

```
A(2, 1)   A(5, 1)   A(8, 1)   A(2, 2)   A(5, 2)
A(8, 2)   A(2, 3)   A(5, 3)   A(8, 3)   A(2, 4)
A(5, 4)   A(8, 4)
```

6.20 Un arreglo matricial A de N × N está en memoria. Escribir un segmento de programa Fortran que imprima los elementos de A por encima de la diagonal, es decir, los elementos A(I, J) para los cuales I < J.

Deseamos imprimir los elementos:

$$\begin{array}{llll} A_{12} & A_{13} & A_{14} \ \ldots & A_{1N} \\ & A_{23} & A_{24} \ \ldots & A_{2N} \\ & & \ldots\ldots\ldots & \\ & & & A_{N-1,N} \end{array}$$

El primer subíndice I denota la fila de los elementos y, por consiguiente, la variaremos desde 1 hasta N – 1. El segundo subíndice J denota la columna y por consiguiente la variaremos desde I+1 hasta N. No podemos usar el ciclo DO implícito anidado

```
WRITE(6, 10) ((A(I, J), J = I + 1, N), I = 1, N - 1)
```

puesto que I+1 y N – 1 no pueden ser parámetros. Sin embargo, podemos usar un ciclo DO junto con un ciclo DO implícito:

```
      NN = N - 1
      DO 100 I = 1, NN
            K = I + 1
            WRITE(6, 10) (A(I, J), J = K, N)
10          FORMAT(1X, 8(3X, F10.2))
100   CONTINUE
```

Problemas suplementarios

ARREGLOS

6.21 Encontrar los errores, si los hay, con cada proposición DIMENSION:

(*a*) DIMENSION, J(8), INTEREST(5, 10),
(*b*) DIMENSION A(4, 0, 3), B(6) C(9, 4)
(*c*) DIMENSION, A(M, N) XYZ(4, −8, 7)

6.22 Determinar el número de elementos en los arreglos NAME, ID, X, Y, de acuerdo con la proposición

```
DIMENSION NAME(5, 300), ID(3, 5, 2), X(50), Y(4, 8)
```

6.23 Utilizando el arreglo ID del problema anterior 6.22, describa la salida si se ejecuta el siguiente par de proposiciones:

```
      WRITE(6, 10) ID
10    FORMAT(1X, 8I8)
```

6.24 Utilizando el arreglo Y del problema 6.22, escribir un par de proposiciones WRITE-FORMAT que impriman las cuatro filas de la proposición Y sobre las primeras cuatro filas de una página de impresión.

6.25 Encontrar la salida para cada segmento de programa:

(*a*)
```
      DIMENSION ID(5)
      DO 100 K = 1, 5
            ID(K) = 2*K
100   CONTINUE
      ID(2) = ID(2) + ID(3)
      ID(5) = ID(5) − ID(4)
      WRITE(6, 10) ID
10    FORMAT(1X, 5I8)
```

(*b*)
```
      INTEGER X(6)
      DO 100 K = 1, 6, 2
            X(K) = 3*K
            X(K + 1) = K + 2
100   CONTINUE
      DO 200 J = 1, 6, 3
            X(J) = X(J) + X(J + 1)
200   CONTINUE
      IF(X(2).LT.7) X(2) = X(3)
      WRITE(6, 20)
20    FORMAT(1X, 6I8)
```

6.26 Supongase que un arreglo lineal A con N elementos está en memoria. Escribir un segmento de programa que:

(*a*) Intercambie A_1 y A_2 solamente si A_1 es mayor que A_2, intercambie A_3 y A_4 solamente si A_3 es mayor que A_4 y así, sucesivamente. (Aquí se supone que N es par.)
(*b*) Permute cíclicamente los valores de A de tal forma que A_2 contenga el valor original de A_1, A_3 contenga el valor original de A_2 y así sucesivamente, con A_1 conteniendo el valor original de A_N.

CICLOS DO IMPLICITOS

6.27 Describa la salida si se ejecuta el par de proposiciones WRITE-FORMAT:

(*a*)
```
      WRITE(6, 10) (A(N), N = 4, 28, 5)
10    FORMAT(1X, 3F15.2)
```

(*b*)
```
      WRITE(6, 20) (J(L), K, N(3, L), L = 2, 13, 3)
20    FORMAT(1X, 4I10)
```

6.28 Describa la salida si se ejecuta el siguiente par de instrucciones WRITE-FORMAT:

(*a*)
```
      WRITE(6, 30) (A(M), M = 1, 5), X, (B(N), N = 1, 7, 3)
30    FORMAT(1X, 4F15.2)
```

(*b*)
```
      WRITE(6, 40) ((A(M, N), M = 1, 5), N = 1, 7, 3)
40    FORMAT(1X, 4F15.2)
```

(*c*)
```
      WRITE(6, 50) ((A(M, N), N = 1, 7, 3), M = 1, 5)
50    FORMAT(1X, 4F15.2)
```

PROGRAMAS

6.29 Una clase de 35 estudiantes hace un examen que se califica de 0 a 100. Escriba un programa Fortran que encuentre: (*a*) el resultado promedio; (*b*) el número de estudiantes que perdieron, es decir, cuyos resultados están por debajo de 60; y (*c*) el número de estudiantes con notas perfectas, es decir, calificados con 100.

6.30 Un estudiante hace siete pruebas (calificadas de 0 a 100), y su nota final es el promedio de las seis notas más altas. Conviniendo en que los siete resultados están perforados en una tarjeta de datos, escribir un segmento de programa Fortran que encuentren la nota final del estudiante (Sugerencia: Reste el resultado más pequeño de la suma de los siete resultados.)

6.31 Un hombre invierte dinero en una cuenta de ahorros cada año durante diez años, que le da 7% de interés anual. Escriba un programa que imprima el dinero que tiene en su cuenta cada año durante veinte años. (Tengase en cuenta que no se hace una inversión adicional en los últimos diez años.)

6.32 Una cadena de almacenes de departamentos tiene seis almacenes y cada almacén tiene los mismos doce departamentos. Las ventas semanales de la cadena están almacenadas en un arreglo SALES de $6 \times 12 \times 7$, tales que SALES(I, J, K) denota las ventas en el almacén I, departamento J en el día K. Escribir un segmento de programa que:

(*a*) Imprima las ventas totales semanales de cada almacén.
(*b*) Imprima las ventas totales semanales de cada departamento.
(*c*) Imprima las ventas totales semanales de la cadena.

(Compare con el problema 6.7.)

6.33 Una cadena de almacenes de departamentos tiene 14 almacenes (numerados de 1 a 14) y cada almacén tiene los mismos 8 departamentos. Semanalmente, cada almacén remite una tarjeta de datos a la oficina central, en la cual se ha perforado el número del almacén en las columnas 1-3 y luego se perforan 8 números que representan las ventas semanales de cada departamento. Escribir un programa Fortran que:

(*a*) Lea las tarjetas de datos en un arreglo SALES, tal que SALES (I, J) contenga las ventas semanales del almacén I, departamento J. (Convenir en que las tarjetas de datos no se ordenan en una forma especial.)
(*b*) Imprimir la venta total de cada departamento.
(*c*) Imprimir el número del almacén con el máximo de ventas.

6.34 Dados los enteros positivos I y J, la secuencia generalizada de Fibonacci está definida por

$$\text{JFIB}(1) = \text{I}, \quad \text{JFIB}(2) = \text{J}, \quad \text{y} \quad \text{JFIB}(\text{K}) = \text{JFIB}(\text{K} - 2) + \text{JFIB}(\text{K} - 1)$$

para $\text{K} > 2$, (ver problema 6.8.) Escribir un programa que lea I y J e imprima los primeros 30 términos de la secuencia, de tal forma que haya tres números por línea.

6.35 Escribir un programa Fortran que:

(*a*) Lea 25 enteros positivos y decida si dos de estos enteros, cualesquiera que ellos sean, sumen 15.
(*b*) Lea 25 enteros positivos y encuentre el segundo número más pequeño y el segundo número más grande.
(*c*) Lea 25 enteros positivos y encuentre el entero par más grande o imprima NO HAY ENTEROS PARES.
(*d*) Lea 25 enteros positivos y cuente los números que son pares y los números que son impares.

6.36 Los resultados de 450 estudiantes de la oficina de exámenes de admisión a la universidad están perforados en tarjetas, uno por tarjeta. (Tales resultados están entre 200 y 800.) Escribir un programa que encuentre el número de resultados entre 200 y 300, entre 300 y 400 y así, sucesivamente.

6.37 Escribir un programa que lea los coeficientes $a_1, a_2, \ldots, a_{n+1}$ del polinomio

$$p(x) = a_1x^n + a_2x^{n+1} + \cdots + a_nx + a_{n+1}$$

los almacene en un arreglo lineal COEF e imprima los valores de $p(x)$ para $x = -5, -4.5, -4, \ldots, 4.5, 5$, esto es, para x desde -5 hasta 5 con incrementos de 0.5. Utilizar el método de Horner para evaluar $p(x)$; esto es,

$$p(x) = ((\ldots((a_1x + a_2)x + a_3)x + \cdots + a_{n-1})x + a_n)x + a_{n+1}$$

6.38 (*Teoría de grupos*) Supóngase que JSET y KSET son arreglos lineales, cada uno de los cuales contiene distintos enteros positivos.

(*a*) Escriba un segmento de programa que almacene en INTER la intersección de JSET y KSET, o sea, aquellos elementos que están tanto en JSET como en KSET.
(*b*) Escriba un segmento de programa que almacene en JUNION la unión de JSET y KSET, o sea, aquellos elementos en JSET o en KSET (o en ambos).

6.39 El inventario de un depósito de artículos de TV está almacenado en memoria en dos arreglos lineales ID y AMOUNT, como sigue. Cada entrada de ID da el número del modelo (número de seis dígitos) de una TV, y la entrada correspondiente en AMOUNT da el número de ese modelo de TV en el depósito. Cada orden de transporte de entrada o de salida remite un paquete de tarjetas de datos (con una tarjeta delimitadora), en el cual el número del modelo está perforado en las columnas 1-8 y la cantidad del artículo transportado es perforado en las columnas 11-20 (el número es positivo si el transporte entra y negativo si sale.) Escribir un programa que de como resultado el inventario después de cada orden de transporte. Se imprimirá también

CANTIDAD INSUFICIENTE DEL MODELO NUMERO XXXXXX

si la orden de salida de un artículo excede el número de artículos almacenados.

6.40 Un arreglo matricial A de N × N está en memoria.

(*a*) Escriba un programa que sume los elementos que están por encima de la diagonal principal, es decir, aquellos elementos A(I, J), tales que I < J.
(*b*) Escriba un programa que sume los elementos que están por debajo de la diagonal principal, es decir, aquellos elementos A(I, J), tales que I > J.

6.41 (*Juego de bolos*) Un arreglo lineal A contiene el número de bolos derribados en cada lanzamiento. Escriba un segmento de programa que almacene en un arreglo B el resultado después de cada línea. Luego, B(10) da el resultado final. Probar el programa con los siguientes datos:

(*a*) 4, 5, 10, 10, 6, 4, 7, 3, 8, 0, 7, 3, 10, 8, 1, 10, 7, 2
(*b*) 6, 4, 7, 2, 10, 7, 3, 10, 5, 4, 3, 0, 10, 10, 8, 2, 7

6.42 (*Método de Sieve*) Escribir un programa que imprima los números primos menores de 400, como sigue:

(*a*) Sea A un arreglo lineal con 400 elementos.
(*b*) Almacene K en A(K); esto es, A(1) = 1, A(2) = 2,..., A(400) = 400.
(*c*) Imprima A(2) y luego haga todos los múltiplos de 2 iguales a cero.
(*d*) Imprimir el siguiente elemento diferente de cero en el arreglo, el cual es A(3), y luego hacer todos los múltiplos de 3 iguales a cero.
(*e*) Continuar el proceso anterior hasta que $A(K) \leq 20 = \sqrt{400}$.
(*f*) Cuando A(K) > 20, imprimir todos los elementos diferentes de cero de A.

Respuestas a problemas suplementarios seleccionados

6.21 (*a*) No debe haber coma después de DIMENSION, ni al final.
(*b*) No debe haber coma después de B(6). Además, cero no puede ser un subíndice.
(*c*) No debe haber coma después de DIMENSION; debe haber coma antes de XYZ; M,N y – 8 no pueden ser subíndices.

6.22 1500, 30, 50, 32

6.23 Puesto que solamente aparece el nombre del arreglo, ID será impreso en el orden de almacenamiento; esto es, ID se imprimirá por columnas de la primera página y luego por columnas de la segunda página con 8 elementos por línea.

6.24
```
      WRITE(6, 20) ((Y(I, J), J = 1, 8), I = 1, 4)
20    FORMAT('1', 8F15.2/1X, 8F15.2/1X, 8F15.2/1X, 8F15.2)
```

6.25 (*a*) Se imprimen los enteros 2, 10, 6, 8, 2 justificados a la derecha en campos de longitud 8.
(*b*) Se imprimen los enteros 6, 9, 9, 20, 15, 7 en campos de longitud 8.

6.26 (*a*)
```
      DO 99 K = 1, N, 2
            IF(A(K).LE.A(K + 1)) GO TO 99
            T = A(K)
            A(K) = A(K + 1)
            A(K + 1) = T
   99 CONTINUE
```

(*b*)
```
      T = A(N)
      NN = N – 1
      DO 88 K = 1, NN
            A(N – K + 1) = A(N – K)
   88 CONTINUE
      A(1) = T
```

6.27 (*a*) A(4), A(9), A(14) en una línea y A(19), A(24) en la siguiente.
(*b*) J(2), K, N(3, 2), J(5) en una línea; K, N(3, 5), J(8), K en la línea siguiente; y N(3, 8), J(11), K, N(3, 11) en la tercera línea.

6.28 (*a*) A(1), A(2), A(3), A(4) en una línea; A(5), X, B(1), B(4) en la siguiente línea; y B(7) en la tercera línea.

(*b*) A(1, 1), A(2, 1), A(3, 1), A(4, 1) en una línea; A(5, 1) A(1, 4), A(2, 4), A(3, 4) en la siguiente línea; A(4,4), A(5, 4), A(1, 7), A(2, 7) en la tercera línea; y A(3, 7), A(4, 7), A(5, 7) en la cuarta línea.
(*c*) A(1, 1), A(1, 4), A(1, 7), A(2, 1) en una línea; A(2, 4), A(2, 7), A(3, 1), A(3, 4) en la línea siguiente; A (3, 7), A(4, 1), A(4, 4), A(4, 7) en la tercera línea; y A(5, 1), A(5, 4), A(5, 7) en la cuarta línea.

1 2 3 4 5 6 7 8 9 10

Capítulo 7

Funciones y subrutinas

7.1 INTRODUCCION

Cuando los programas llegan a ser demasiado complejos, surgen entonces varios problemas inmediatos:

1. Los algoritmos de solución se hacen más difíciles y, por lo tanto, complicados en el diseño.
2. Aunque se conozca un algoritmo, su implementación en Fortran llega a ser dificultuosa, porque la magnitud del programa es extensa.
3. Cuando los programas se hacen más largos y más complejos, la depuración se hace entonces más difícil; además, son más propensos a errores, y estas equivocaciones son más difíciles de aislar.
4. Se requiere más y más documentación para hacer que el programa sea más comprensible para las personas que lo leerán y utilizarán.
5. Se requieren tareas similares en varias partes de un programa. Si se implementan individualmente, hacen que el problema complejo sea aún más confuso.

Estos programas se pueden mejorar con el uso de subprogramas. Un *subprograma* es un programa completo e independiente que puede utilizarse (o: invocarse o llamarse) en el programa principal o en otros subprogramas. Un subprograma recibe valores (llamados *argumentos*) de un llamado (desde el origen) de programa, ejecuta cálculos y luego devuelve (RETURNs) el resultado o resultados, al llamado de programa.

En el capítulo 2, discutimos algunas funciones de biblioteca, tales como ABS, SQRT, EXP, ALOG, etc. Estas funciones realmente son subprogramas. Se escribieron e incorporaron al repertorio del Fortran, porque se utilizan en muchísimas situaciones. Es una gran ventaja el hecho de que los compiladores provean estas funciones internas de biblioteca, para que cada usuario no tenga que escribir estos subprogramas. Obviamente, la librería Fortran no puede contener todas las funciones o subprogramas utilizados en todas las áreas de aplicación. Sin embargo, es posible que los usuarios escriban sus propios subprogramas cada vez que la necesidad así lo indique. Este capítulo está dedicado a la discusión de cómo se preparan y utilizan tales subprogramas.

Recordemos que el arreglo del paquete Fortran se mostró en la figura 1-3 de la sección 1.3. El paquete Fortran constará del programa principal, con todos los subprogramas. Usualmente, se coloca primero el programa principal y a continuación los subprogramas. Esto se ilustra en la figura 7-1, donde el paquete Fortran tiene dos subprogramas:

Los subprogramas se clasifican en dos categorías básicas: subprogramas FUNCTION y subprogramas SUBROUTINE. Las semejanzas y diferencias de estos dos tipos de subprogramas se examinarán en las próximas secciones. Aunque ambos tipos de subprogramas pueden recibir uno o más valores del llamado de programa, el subprograma FUNCTION usualmente devuelve un solo resultado, mientras que el subprograma SUBROUTINE generalmente devuelve más de un resultado.

7.2 SUBPROGRAMAS FUNCTION

Observamos que cada una de las funciones de biblioteca: ABS, SQRT, SIN, ALOG, etc., calcula un valor cada vez que se da el argumento apropiado. Supongamos, por ejemplo que,

```
SQRT(A**2 + B**2)
```

aparece en una expresión aritmética. Cuando ésta se ejecuta, el computador:

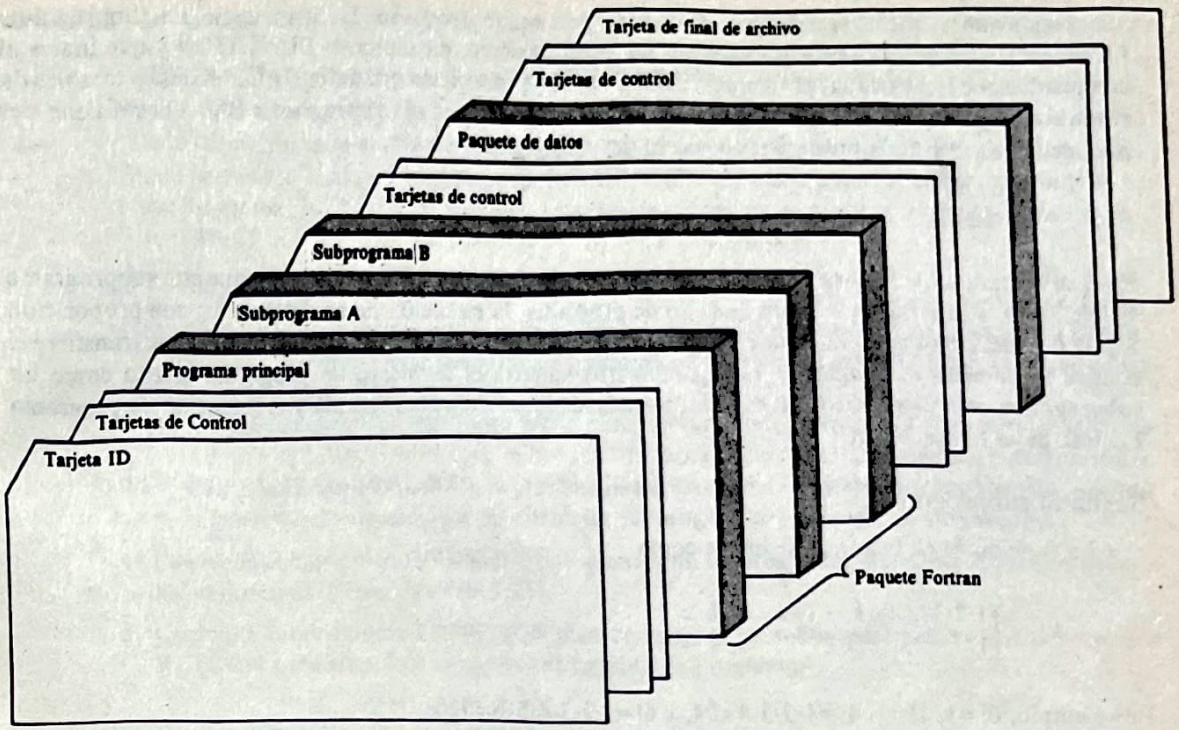


Fig. 7-1 El paquete Fortran.

1. Llama al subprograma SQRT.
2. Calcula el valor real de $A^2 + B^2$ y utiliza este valor como argumento del subprograma.
3. El subprograma calcula la raíz cuadrada del argumento dado (el cual es $A^2 + B^2$) y asigna este valor real a la variable SQRT en el subprograma.
4. Envía el valor de SQRT del subprograma al llamado de programa y su valor se sustituye entonces por SQRT(A**2 + B**2).

Obsérvese que este subprograma tiene un valor asociado con su nombre SQRT. Los subprogramas de esta clase se denominan subprogramas FUNCTION.

Presentamos ahora dos ejemplos de subprogramas FUNCTION. Nos referiremos a ellos más adelante, cuando se hayan enumerado las reglas sobre escritura de subprogramas.

Primer subprograma

Consideremos el problema de hallar el mayor de tres números A, B y C. Suponemos que A, B y C están en la memoria y que BIG es la variable real que utilizaremos para almacenar el valor mayor; la figura 7-2(*a*) da un segmento de programa Fortran, que termina una vez calculado BIG.

```
BIG = A
IF(BIG.LT.B) BIG = B
IF(BIG.LT.C) BIG = C
STOP
END
```

(*a*)

```
FUNCTION BIG(A, B, C)
BIG = A
IF(BIG.LT.B) BIG = B
IF(BIG.LT.C) BIG = C
RETURN
END
```

(*b*)

Supongamos que deseamos escribir este segmento de programa Fortran como un subprograma FUNCTION. La primera proposición debe ser la proposición que define la FUNCTION y que indica al compilador que éste es un subprograma FUNCTION. La proposición que define la FUNCTION también da el nombre del subprograma y sus parámetros. Llamaremos nuestro subprograma BIG, el cual tiene tres parámetros A, B y C; la primera proposición del subprograma será

```
FUNCTION BIG(A, B, C)
```

En el subprograma, se supone que estos parámetros están definidos. Más aún, puesto que este subprograma FUNCTION se utilizará por algún llamado de programa, la ejecución no se detiene por una proposición STOP después del cálculo, sino que el subprograma contendrá una proposición RETURN, que transfiere el control al llamado de programa. La figura 7-2(*b*) muestra el segmento de programa escrito como un subprograma completo FUNCTION. Obsérvese la similitud del subprograma y el segmento de programa Fortran de la figura 7-2(*a*).

Segundo subprograma

La función $k!$ (k factorial) se define como

$$k! = \begin{cases} 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (k-1) \cdot k, & \text{si } k \geq 1 \\ 1 & \text{si } k = 0 \end{cases}$$

Por ejemplo, $0! = 1$, $1! = 1$, $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$, y $6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$.

Consideremos el problema de calcular $k!$ para cualquier entero no negativo k. Suponemos que k está en la memoria y que IFACT es la variable entera que se utiliza para almacener $k!$; la figura 7-3(*a*) da un segmento de programa Fortran que termina después de calcular IFACT. Por otra parte, la figura 7-3(*b*) muestra cómo IFACT puede obtenerse desde un subrprograma FUNCTION. Obsérvese que hay dos proposiciones RETURN en correspondencia con las dos proposiciones STOP.

```
      IFACT = 1
      IF(K.EQ.0) STOP
      DO 10 J = 1, K
            IFACT = IFACT*J
10    CONTINUE
      STOP
      END
```

(*a*)

```
      FUNCTION IFACT(K)
C
C     CALCULO DE K FACTORIAL
C
      IFACT = 1
      IF(K.EQ.0) RETURN
      DO 10 J = 1, K
            IFACT = IFACT*J
10    CONTINUE
      RETURN
      END
```

(*b*)

Fig. 7-3

Ahora resumimos los puntos principales que deben tenerse en cuenta en la escritura de subprogramas FUNCTION:

1. La primera proposición del subprograma debe ser la proposición que define la FUNCTION y que tiene la siguiente forma:

   FUNCTION NOMBRE(param 1, param 2,..., param *n*)

   Esto es, a la palabra FUNCTION le sigue el NOMBRE de la función y a éste le siguen los parámetros, que están separados por comas y encerrados entre paréntesis. Estos parámetros deben ser nombres de variables no subindicadas (o nombres de arreglos, véase sección 7.6), los que se supone están definidos completamente, al ser utilizados en el subprograma.

2. Para asignar un nombre al subprograma FUNCTION, se deben seguir las reglas usuales para nombres de variables; es decir, debe contener uno o seis caracteres alfanuméricos en donde el primero debe ser alfabético. El NOMBRE del subprograma FUNCTION (sin incluir sus parámetros) también debe definirse de alguna manera en el subprograma (es decir, debe aparecer al lado izquierdo de cualquier proposición de asignación o en una proposición de entrada), antes de que se ejecute cualquier proposición RETURN. (De tal manera que al asignar un nombre al subprograma FUNCTION también se sigan las convenciones de tipo para nombres de variables.)
3. Un subprograma es un programa completo e independiente:
   (*a*) Es *completo*, porque contiene las declaraciones de tipo necesarias (tales como, INTEGER, REAL, DIMENSION, etc.), utiliza funciones de biblioteca y otros subprogramas, así como la proposición END.
   (*b*) Es *independiente* porque todos los nombres de variables (excepto los parámetros) y todos los rótulos de proposiciones que se utilizan son locales dentro del subprograma y desconocidos fuera de él. De esta manera, se pueden utilizar los mismos nombres de variables y los mismos números de proposiciones en diferentes subprogramas y llamado de subprogramas.
4. La comunicación entre un llamado de programa y un subprograma, solamente se hace a través de los parámetros. (Véase sección 7.3.)
5. Cualquier subprograma FUNCTION debe contener al menos un parámetro y una proposición RETURN para transferir el control al llamado de programa.

7.3 LLAMADO DE SUBPROGRAMAS FUNCTION

Un subprograma FUNCTION se *invoca* (o: se *llama*) de la misma manera que las funciones de biblioteca. O sea, que en aquellos lugares donde se desean conocer los valores de la función entonces, se escribe:

NOMBRE(arg 1, arg 2,..., arg*n*)

Esto es, se escribe el NOMBRE de la función seguido por expresiones adecuadas para sus argumentos, los cuales se encierran entre paréntesis y se separan por medio de comas.

No se requiere que los argumentos de la proposición de llamado sean los mismos que los correspondientes parámetros en la proposición que define la FUNCTION. En efecto, como en las funciones de biblioteca, los argumentos pueden ser constantes, variables simples, expresiones aritméticas y aun nombres de funciones de biblioteca u otros subprogramas. El hecho más importante que debe recordarse es que los argumentos deben corresponder uno a uno con el *orden* y el *tipo de dato* de los parámetros correspondientes a la proposición que define la FUNCTION. Por ejemplo, consideremos el subprograma FUNCTION BIG mencionado anteriormente y cuya proposición de definición es

```
FUNCTION BIG(A, B, C)
```

Obsérvese que se requieren tres argumentos reales para llamar a BIG, de manera que las siguientes proposiciones son todas correctas en cuanto a sintaxis se refiere:

```
GRADE = BIG(T1, T2, T3)
GRADE = FINAL/3.0 + (2.0/3.0)*BIG(T1, 60.0, SQRT(A))
GRADE = 0.5*(FINAL + BIG(S(I, 1), S(I, 2), Z(I))
```

Se ilustra en seguida la correspondencia uno a uno entre los argumentos de la segunda proposición y los parámetros del subprograma:

```
Lista de argumentos:    T1,    60.0,    SQRT(A)
                          ↓       ↓        ↓
Lista de parámetros:    A,     B,       C
```

Hay un punto sutil que debe comprenderse cuando se utilizan subprogramas. Los parámetros que se utilizan en la proposición que define la FUNCTION se denominan *variables ficticias*, porque cuando ellas se establecen no ocupan localizaciones de almacenamiento. Específicamente, las direcciones de los argumentos correspondientes en el llamado de programa se utilizan en la ejecución del subprograma. Por ejemplo, en el llamado de programa

```
GRADE = BIG(T1 + T2, C, 60.0)
```

las direcciones de los argumentos T1+T2, C y 60.0 se utilizan en lugar de los parámetros A, B, C en la ejecución del subprograma BIG.

Nota. Obsérvese que utilizamos el término *parámetros* para las variables que aparecen en la proposición que define el subprograma y el término *argumentos* para los valores correspondientes al llamado de programa.

EJEMPLO 7.1

(*a*) El coeficiente binomial $\binom{n}{i}$ para enteros no negativos n e i (donde $i \leq n$) se define como

$$\binom{n}{i} = \frac{n!}{(n-i)!i!}; \text{ por ejemplo, } \binom{8}{3} = \frac{8!}{5!3!} = \frac{1\cdot2\cdot3\cdot4\cdot5\cdot6\cdot7\cdot8}{1\cdot2\cdot3\cdot4\cdot5\cdot1\cdot2\cdot3} = \frac{6\cdot7\cdot8}{1\cdot2\cdot3} = 56$$

Escriba un programa que lea enteros no negativos N e I (donde I $\leq$ N) e imprima el valor $\binom{N}{I}$.

Si se utiliza el subprograma IFACT de la sección anterior, el siguiente programa Fortran nos da el coeficiente binomial requerido.

```
C       PROGRAMA PARA CALCULAR COEFICIENTE BINOMIAL
        INTEGER UP, DOWN
        READ(5, 10) N, I
   10   FORMAT(2I8)
        UP = IFACT(N)
        DOWN = IFACT(N - I)*IFACT(I)
        IBINQ = UP/DOWN
        WRITE(6, 20) N, I, IBINQ
   20   FORMAT(6X, 'N = ', I5, 5X, 'I = ', I5, 5X,
       1        'COEFICIENTE BINOMIAL = ', I8)
        STOP
        END
```

Obsérvese que el uso del subprograma hace que el programa prinicipal sea sencillo y claro. Por otra parte, si $n!$, $(n-i)!$ e $i!$ se calcularan separadamente en el programa principal, el programa sería extenso, menos comprensible y más propenso a error. Obsérvese también que IBINQ se puede calcular por medio de la proposición:

```
IBINQ = IFACT(N)/(IFACT(N - I)*IFACT(I))
```

Sin embargo, las expresiones más extensas y más complejas están más propensas a error. Cuando se escriben expresiones complicadas, generalmente es deseable separarlas en varias expresiones más imples.

Nota. Ignoramos el hecho de que $n!$ puede ocasionar sobrecupo para valores de n muy grandes. Más adelante se discutirán otras formas de calcular $\binom{n}{i}$.

(*b*) Supongamos que a un curso de 25 estudiantes se practican tres exámenes parciales y un examen final. Los resultados, junto con los números de identificación ID de los estudiantes se perforan en tarjetas, a razón de una por cada estudiante. El grado numérico para el curso se calcula como el promedio del examen final y el resultado más alto de los tres exámenes parciales.

El siguiente programa, que utiliza el subprograma BIG, calcula el grado numérico para cada estudiante:

```
      WRITE(6, 10)
   10 FORMAT('1', 4X, 'ID', 6X, 'RESUL 1', 2X, 'RESUL 2',
     1          2X, 'RESUL 3', 6X, 'FINAL', 5X, 'GRADO')
      DO 500 I = 1, 25
            READ(5, 20) ID, T1, T2, T3, FINAL
            GRADO = 0.5*(FINAL + BIG(T1, T2, T3))
            WRITE(6, 30) ID, T1, T2, T3, FINAL, GRADO
  500 CONTINUE
   20 FORMAT(I10, 4(F6.2, 4X))
   30 FORMAT(1X, I6, 5(3X, F6.2))
      STOP
      END
```

Nota. Cualquier subprograma puede llamar a cualquier otro subprograma, pero no puede llamarse a sí mismo. Más generalmente, los llamados de programas no pueden formar un ciclo; por ejemplo, no se puede tener una situación donde:

Subprograma A llama al subprograma B,

Subprograma B llama al subprograma C,

Subprograma C llama al subprograma D, y

Subprograma D llama al subprograma A.

Una manera de garantizar que esto no suceda, es colocar un subprograma antes de cualquier otro subprograma que él invoque; por ejemplo, si los subprogramas X y Y llaman al subprograma Z, entonces Z se coloca después de X y Y en el paquete Fortran.

7.4 SUBPROGRAMAS FUNCTION PARA EL CALCULO DE VARIOS VALORES

Hasta aquí, todos nuestros subprogramas FUNCTION calculaban un valor y éste era asignado al NOMBRE de la función en el subprograma. Pregunta: ¿Puede un subprograma FUNCTION calcular varios valores? La respuesta es "Si" (aunque el subprograma SUBROUTINE que se discute en la sección 7.7 se utiliza generalmente en tal situación).

Recordemos que la comunicación entre un llamado de programa y un subprograma solamente se hace por la vía de los argumentos y los parámetros. Por consiguiente, cada vez que un subprograma FUNCTION se utiliza para calcular varios valores, se asigna un valor al NOMBRE de la función y los otros se asignan a las variables ficticias (que también se transmiten al llamado de programa.)

EJEMPLO 7.2

Supongamos que deseamos escribir un subprograma FUNCTION para calcular el número mayor y la suma de tres números A, B y C. Llamamos al subprograma BBB, donde BBB se utiliza para almacenar el número mayor. Este subprograma es similar al subprograma BIG de la figura 7-2, excepto que BBB debe tener cuatro parámetros, digamos SUM, que se usan para transferir la suma de A, B y C al llamado de programa. El subprograma BBB es como sigue:

```
FUNCTION BBB(A, B, C, SUM)
SUM = A + B + C
BBB = A
IF(BBB.LT.B) BBB = B
IF(BBB.LT.C) BBB = C
RETURN
END
```

Supongamos que el llamado de programa tiene la proposición

```
RESULT = BBB(X, Y, Z, TOTAL)
```

Después de que se ejecuta esta proposición, RESULT contendrá el mayor de los numeros X, Y y Z y TOTAL contendrá la suma. TOTAL estará entonces disponible para cálculos. Por ejemplo, si

```
PROM = TOTAL/3.0
```

se ejecuta un tiempo más tarde, entonces PROM contendrá el promedio de X, Y y Z.

7.5 ARREGLOS Y SUBPROGRAMAS FUNCTION, DIMENSION VARIABLE

También se puede utilizar un arreglo como parámetros de un subprograma FUNCTION. En tal caso, su argumento correspondiente en la proposición de llamado debe ser también un arreglo. Sin embargo, el arreglo en el subprograma debe declararse por medio de una proposición DIMENSION. Además, sus dimensiones no pueden exceder a las de su argumento correspondiente.

La figura 7-4(*a*) es un subprograma llamado BIGG, que encuentra el mayor elemento de un arreglo lineal A de 25 elementos y la figura 7-4(*b*) es un típico llamado de programa.

```
      FUNCTION BIGG(A)
      DIMENSION A(25)
      BIGG = A(1)
      DO 10 K = 2, 25
            IF(BIGG.LT.A(K)) BIGG = A(K)
   10 CONTINUE
      RETURN
      END
```

(*a*)

```
      DIMENSION X(25)
      REAL LARGE
      READ(5, 10) X
   10 FORMAT(5(F8.2, 2X))
      LARGE = BIGG(X)
      WRITE(6, 20) LARGE
   20 FORMAT('0', 'EL MAYOR VALOR ES', 2X, F8.2)
      STOP
      END
```

(*b*)

Fig. 7-4

El anterior subprograma BIGG encontrará el mayor elemento de un arreglo lineal que tiene exactamente 25 elementos, lo que obviamente, limita su uso. Para permitir más generalidad, Fortran nos permite uitilizar *dimensiones variables* (es decir, dimensiones ajustables) en los subprogramas. Esto lo discutimos enseguida.

Dimensión variable. Recordemos que en cualquier programa, el subíndice de un nombre de arreglo en una proposición DIMENSION debe ser una constante entera y no una variable. Específicamente, supongamos que en un subprograma una proposición DIMENSION declara que una variable A es un arreglo y que A es un parámetro del subprograma. Entonces, en la proposición DIMENSION se puede suministrar una variable entera que sea un subíndice de A y parámetro del subprograma. Esto se ilustra en el siguiente subprograma FUNCTION que encuentra el mayor elemento de un arreglo A con N elementos.

```
      FUNCTION BIGMM(A, N)
      DIMENSION A(N)
      BIGMM = A(1)
      DO 10 K = 1, N
            IF(BIGMM.LT.A(K)) BIGMM = A(K)
   10 CONTINUE
      RETURN
      END
```

Hacemos hincapié en que el uso de la dimensión variable se permite solamente en los subprogramas, pero nunca en el programa principal.

Nota. El valor de un parámetro N utilizado en una proposición DIMENSION variable no debe exceder la magnitud del arreglo correspondiente en el llamado de programa.

EJEMPLO 7.3

(*a*) Escriba un subprograma FUNCTION que calcule la suma de los elementos de un arreglo lineal que tiene N elementos. Aquí, tanto A como N son parámetros:

```
      FUNCTION SUM(A, N)
      DIMENSION A(N)
      SUM = 0.0
      DO 99 K = 1, N
            SUM = SUM + A(K)
   99 CONTINUE
      RETURN
      END
```

(*b*) Supongamos que X y Y son arreglos lineales que están en la memoria y que se declaran como

```
DIMENSION X(100), Y(200)
```

Utilizamos (*a*) para encontrar los promedios:

$$(X_1 + X_2 + \cdots + X_{25})/25$$
$$(Y_1 + Y_2 + \cdots + Y_M)/M$$

Tenemos entonces:

```
AVEX = SUM(X, 25)/25.0
AVEY = SUM(Y, M)/FLOAT(M)
```

(Suponemos que $M \leq 200$.)

7.6 PROPOSICION QUE DEFINE FUNCIONES ARITMETICAS

Supongamos que deseamos evaluar la función cuadrática

$$g(x) = x^2 - 5x + 2$$

para $x = 1, 2, \ldots, 20$. Utilizando un subprograma FUNCTION, tenemos:

Llamado de programa	Subprograma
`DO 100 J = 1, 20`	`FUNCTION G(X)`
`  VALUE = G(FLOAT(J))`	`G = X*X - 5.0*X + 2.0`
`  WRITE(6, 10) J, VALUE`	`RETURN`
`100 CONTINUE`	`END`
`10 FORMAT(1X, I3, 3X, F8.2)`	
`STOP`	
`END`	

Obsérvese que este subprograma FUNCTION consiste en una sola proposición aritmética (además de la proposición que define la FUNCTION y las proposiciones RETURN y END). Podemos simplificar esta clase de subprogramas FUNCTION, escribiendo simplemente la proposición

```
G(X) = X*X - 5.0*X + 2.0
```

al comienzo del llamado de programa, como sigue:

```
C
C       DEFINIMOS LA FUNCION G
C
        G(X) = X*X - 5.0*X + 2.0
C
C       CALCULO DE LOS VALORES DE LA FUNCION
C
        DO 100 J = 1, 20
              VALUE = G(FLOAT(J))
              WRITE(6, 10) J, VALUE
  100   CONTINUE
   10   FORMAT(1X, I3, 3X, F8.2)
        STOP
        END
```

Esta función G se denomina *proposición que define una función aritmética.*

Las reglas para el uso de la proposición que define funciones son:

1. Una proposición que define una función se declara utilizando la siguiente forma:

   NOMBRE(param1, param2,..., param*n*) = expresión aritmética

   Los parámetros deben ser solamente variables no subindicadas y las expresiones aritméticas no deben contener variables con subíndice. Sin embargo, la expresión aritmética puede contener funciones de biblioteca, subprogramas FUNCTION y otras proposiciones que definen funciones (al propocionarse, ellas deben estar definidas previamente). Aunque algunos compiladores permiten que las proposiciones de definición aparezcan en cualquier parte del programa antes de su uso, muchos compiladores requieren que ellos se coloquen al principio del programa después de cualquier proposición de declaración de tipo, pero antes de cualquier proposición ejecutable.

2. La asignación de nombres a la función y a sus parámetros sigue las mismas reglas que para los subprogramas FUNCTION. Como en los subprogramas FUNCTION, los parámetros son variables ficticias, y no se asignan a localizaciones de almacenamiento. Por lo tanto, ellas son locales para la proposición.

3. Las reglas para el llamado de una proposición que define funciones son también las mismas que para los subprogramas FUNCTION. Esto es, en aquellos sitios del programa donde se desea el valor de la función, simplemente se escribe el nombre de la proposición que define la función con argumentos apropiados. Los argumentos pueden ser expresiones aritméticas siempre y cuando ellas coincidan uno a uno en orden y tipo con los parámetros que aparezcan en la definición de la proposición. Por ejemplo, si suponemos que la proposición anterior que define la función G(X) se ha definido en el programa, se puede escribir

   ```
   PROM = (G(A + SQRT(B)) + DEP)/2.0
   ```

   donde PROM viene a ser el promedio de DEP y G, calculados en A + SQRT(B). En este caso, A + SQRT(B) es el argumento que corresponde al parámetro X en la proposición que define la función.

4. Un punto importante es que una proposición que define una función es interna para el programa (o subprograma) en que aparezca; por lo tanto, ella no puede llamarse por cualquier otro programa.

Puesto que una proposición que define una función es interna para su programa, ella tiene un grado de libertad, que no tienen los subprogramas FUNCTION. Esto quiere decir que las variables que no aparecen en

la lista de parámetros, pueden aparecer en la proposición. Por ejemplo, la siguiente proposición que define una función aritmética es correcta en cuanto a sintaxis se refiere:

```
F(X, Y) = A*X**2 + B*X*Y + C*Y**2
```

En este caso, las variables X e Y son ficticias; por lo tanto, se puede utilizar a X y a Y como nombres de variables en otra parte del programa. Sin embargo, las variables A, B y C no son ficticias, puesto que no están en la lista de parámetros; se aceptará que ellas se definan cuando se llame a la función. Por ejemplo,

```
F(1.0, 2.0)
```

calculará la función para los valores actuales de A, B y C.

EJEMPLO 7.4

Se da un paquete de N tarjetas, en cada una de las cuales se perforan tres números reales. Tales números son los coeficientes de la función cuadrática

$$F(X) = AX^2 + BX + C$$

Escriba un programa Fortran que evalúe para cada terna A, B y C los valores de la función para X = −5, −4, ..., 5.

Obsérvese que hay 11 valores de X. Utilizaremos un ciclo DO con índice I = 1, 2, ..., 11. La relación entre el índice I y los valores de X puede mostrarse como sigue:

I:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11
X:	−5,	−4,	−3,	−2,	−1,	0,	1,	2,	3,	4,	5

Obsérvese que X e I están ligados por I − X = 6. Por lo tanto,

```
X = -6 + I
```

El programa requerido es entonces el siguiente:

```
C
C       EJEMPLO SOBRE UTILIZACION DE PROPOSICION QUE DEFINE UNA FUNCION
C
        F(X) = A*X*X + B*X + C
        READ(5, 5) N
      5 FORMAT(I3)
        DO 100 J = 1, N
              READ(5, 10) A, B, C
     10       FORMAT(3F5.1)
              WRITE(6, 15) A, B, C
     15       FORMAT('1', 'A = ', F5.1/1X, 'B = ', F5.1/1X, 'C = ', F5.1)
C
C             EVALUACION DE LA FUNCION
C
              DO 200 I = 1, 11
                    Y = F(-6.0 + FLOAT(I))
                    WRITE(6, 20) I, Y
    200       CONTINUE
     20       FORMAT(1X, 3I, 2X, F8.2)
    100 CONTINUE
        STOP
        END
```

7.7 SUBRUTINAS

Hay otro tipo de subprograma denominado subprograma SUBROUTINE. Recordamos primero que al nombre de un subprograma FUNCTION se asigna siempre un valor en el subprograma. Esto no es necesario

para los subprogramas SUBROUTINE. A pesar de esta diferencia y de su implicación, las reglas y restricciones que gobiernan la escritura de subprogramas SUBROUTINE son análogas a las que se refieren a los subprogramas FUNCTION. Notamos que generalmente se utiliza una SUBROUTINA en lugar de un subprograma FUNCTION, cuando se tienen que calcular varios valores. Más aún, además de calcular valores, se utilizan subprogramas SUBROUTINE para ejecutar tareas (por ejemplo, intercambiar valores, etc.).

Como en los subprogramas FUNCTION, los subprogramas SUBROUTINE primero deben declararse, utilizando una proposición que defina a SUBROUTINE. La declaración tiene la siguiente forma general:

SUBROUTINE NOMBRE (param1, param2,..., param*n*)

Puesto que al NOMBRE de la subrutina no se asigna un valor en el subprograma, la pregunta del tipo no se sucede. (Sin embargo, los nombres de las subrutinas no pueden excederse de seis caracteres.)

Como en cualquier subprograma, una SUBRUTINA es también un programa completo e independiente. Por lo tanto, puede contener todas las características de cualquier subprograma: declaraciones de tipo, proposiciones DIMENSION, otros subprogramas FUNCTION y/o SUBROUTINE, la proposición END y así, sucesivamente. Además, la comunicación entre un llamado de programa y el subprograma SUBROUTINE se transmite solamente por medio de los parámetros; por lo tanto, en la SUBRUTINA los nombres de variables (diferentes de los parámetros) y rótulos de proposiciones son todos locales para ese subprograma. (La forma como se llama a una SUBRUTINA y la manera como se transmiten los valores por medio de los parámetros se discutirá más tarde en esta sección.)

Damos ahora dos ejemplos de subprogramas SUBROUTINE. El primero desarrollará una tarea y el segundo calculará valores.

1. Si se asume que X y Y están definidas, el siguiente segmento de programa intercambia los valores de X y Y:

```
T = X
X = Y
Y = T
```

Podemos escribir un subprograma SUBROUTINE llamado INTCHG para ejecutar el intercambio:

```
SUBROUTINE INTCHG(X, Y)
T = X
X = Y
Y = T
RETURN
END
```

Obsérvese primero que al nombre INTCHG no se asigna un valor en el subprograma. (De esta manera, comenzar el nombre con la letra I no tiene importancia.) Si se supone que los valores intercambiados se utilizarán más adelante en el llamado de programa, tenemos la proposición RETURN que transfiere el control al llamado de programa.

2. Considere de nuevo el problema de escribir un programa que calcule el mayor y la suma de tres números A, B y C. Esto se puede hacer con un subprograma SUBROUTINE, donde utilizamos los argumentos para transmitir los valores calculados al llamado de programa. Dicha SUBRUTINA es la siguiente:

```
SUBROUTINE LARGE(A, B, C, BIG, SUM)
SUM=A+B+C
BIG=A
IF(BIG.LT.B) BIG=B
IF(BIG.LT.C) BIB=C
RETURN
END
```

Obsérvese la semejanza y la diferencia entre este subprograma y el subprograma FUNCTION del ejemplo 7.2. En este caso el nombre LARGE no tiene valor; por lo tanto BIG, el mayor de los

valores A, B y C, deberá estar en la lista de parámetros de manera que él pueda transmitirse al llamado de programa.

Ahora nos preguntamos cómo se invocan los subprogramas SUBROUTINE. Como se dijo anteriormente, a diferencia de un subprograma FUNCTION, el nombre de un subprograma SUBROUTINE no toma un valor; por esta razón, se requiere una nueva clase de proposición —la proposición CALL—. La siguiente proposición de llamado invoca (llama) a la subrutina INTCHG

```
CALL INTCHG(A(3), T)
```

El efecto neto de esta proposición en el llamado de programa es que los valores de A(3) y T sean intercambiados.

Advertimos que después de que se invoca INTCHG, las direcciones de A(3) y T se transfieren al subprograma, en donde son sustituidas por las variables ficticias X y Y. El hecho de que el nombre T se utilice en el llamado de programa y también en la subrutina INTCHG no causa complicaciones, porque la variable T es local para el subprograma SUBROUTINE y solamente la dirección del argumento T del llamado (no el nombre T) se utiliza en la ejecución de la SUBRUTINA.

La siguiente proposición de llamado es también correcta en cuanto a sintaxis se refiere:

```
CALL LARGE(U + V, W – SQRT(V), U*U, T, S)
```

El efecto neto de esta proposición en el llamado de programa es el de que S contendrá la suma de U + V, W – SQRT(V) y U*U y T contendrá el mayor de los tres valores.

Enfatizamos de nuevo, como en el caso de los subprogramas FUNCTION, que los argumentos en la proposición de llamado deben coincidir de manera uno a uno con el orden y el tipo de los parámetros en la proposición que define la proposición SUBROUTINE. Ilustramos este hecho, utilizando la anterior proposición CALL:

Lista de argumentos:	U + V,	W – SQRT(V),	U*U,	T,	S
	↓	↓	↓	↓	↓
Lista de parámetros:	A,	B,	C,	BIG,	SUM

EJEMPLO 7.5

Supongamos que A es un arreglo lineal que contiene al menos N elementos. Escriba un subprograma SUBROUTINE que:

1. Encuentre el valor mayor entre los primeros K elementos,

   $$A(1), A(2), \ldots, A(K).$$

2. Calcule la suma de los primeros K elementos,

   $$A(1) + A(2) + \cdots + A(K)$$

La siguiente subrutina SEEK lleva a cabo lo anterior:

```
      SUBROUTINE SEEK(A, N, K, HIGH, SUM)
      DIMENSION A(N)
      SUM = 0.0
      HIGH = A(1)
      DO 100 J = 1, K
            SUM = SUM + A(J)
            IF(HIGH.LT.A(J)) HIGH = A(J)
  100 CONTINUE
      RETURN
      END
```

Como se vio en el ejemplo anterior, los nombres de arreglos pueden utilizarse como parámetros en un subprograma SUBROUTINE (cuando ellos se proporcionan a un subprograma, también aparecen en una

proposición DIMENSION) y pueden tener dimensión variable. Esto es, las reglas que gobiernan los arreglos en los subprogramas SUBROUTINE son similares a las de los subprogramas FUNCTION.

EJEMPLO 7.6

Supongamos que un arreglo A con N elementos se almacena en orden ascendente; o sea, $A(1) \leq A(2) \leq \ldots \leq AN$). Escriba un subprograma SUBROUTINE para rearreglar los elementos del arreglo en orden descendente, esto es, $A(1) \geq A(2) \geq \ldots \geq A(N)$.

Podemos, obviamente, ordenar el arreglo en orden descendente. Sin embargo, esto sería muy ineficaz puesto que sabemos que A está en orden ascendente. Una forma simple de llevar a cabo nuestra tarea es invertir el arreglo; esto es, intercambiar A(1) y A(N), intercambiar A(2) y A(N−1) y así, sucesivamente. Llamamos a esta subrutina INVERT, que a su vez llama a la subrutina INTCHG:

```
      SUBROUTINE INVERT(A, N)
      DIMENSION A(N)
      NN = N/2
      K = N + 1
      DO 20 I = 1, NN
            CALL INTCHG(A(I), A(K − I))
   20 CONTINUE
      RETURN
      END
```

(Obsérvese la proposición DO. ¿Puede el lector indicar la razón por la cual el ciclo DO va solamente hasta N/2 en lugar de ir hasta N?)

El empleo de subprogramas permite convertir problemas grandes y complejos en programas unitarios más pequeños. Cada programa unitario será más sencillo para implementar y depurar. Más aún, los programas unitarios también se pueden ejecutar por diferentes personas de un equipo. Por consiguiente, la tarea de programar en grande puede ejecutarse en paralelo en lugar de hacerse en orden secuencial.

7.8 SUBRUTINA CONTRA FUNCTION

Generalmente, los subprogramas FUNCTION se utilizan para calcular un solo valor, mientras que los subprogramas SUBROUTINE se utilizan para calcular varios valores o para ejecutar tareas (por ejemplo, intercambiar dos valores). Observemos que cualquier cosa hecha por un subprograma FUNCTION también la puede llevar a cabo un subprograma SUBROUTINE. Recíprocamente, muchas de las tareas ejecutadas por los subprogramas SUBROUTINE también las pueden realizar los subprogramas FUNCTION. Por ejemplo, escribimos un subprograma FUNCTION para intercambiar los valores de X y Y:

```
FUNCTION EXCHG(X, Y)
T = X
X = Y
Y = T
EXCHG = 0.0
RETURN
END
```

Obsérvese que al nombre de la FUNCION se asigna arbitrariamente un valor, ya que él se definirá en el subprograma. Puesto que la comunicación se hace a través de los parámetros, los valores de X y de Y se intercambiarán en el programa principal. Sin embargo, la organización es un poco artificial y forzada. Las siguientes son las principales diferencias entre un subprograma SUBROUTINE y un subprograma FUNCTION:

1. Al NOMBRE de una SUBRUTINA no se le asocia ningún valor, mientras que al NOMBRE de un subprograma FUNCTION sí se le asocia un valor numérico o lógico. Por lo tanto, el NOMBRE de un subprograma FUNCTION debe seguir las reglas en cuanto a convención de tipo y debe estar definido en el subprograma.
2. Una SUBRUTINA solamente se puede invocar por una proposición especial de llamado —la proposición CALL—. El NOMBRE de un subprograma FUNCTION puede utilizarse de la

misma manera que las funciones de biblioteca, es decir, en proposiciones de asignación aritmética, etc.

3. Un subprograma FUNCTION debe tener al menos un argumento, mientras que una SUBRUTINA no necesariamente debe tenerlo. (Véase problema 7.8.)
4. Puesto que un subprograma FUNCTION calcula al menos un valor, debe contener una proposición RETURN. Un subprograma SUBROUTINE no necesita contener una proposición de retorno. (Véase problema 7.8.)

Problemas resueltos

SUBPROGRAMAS

7.1 Encuentre los errores, si los hay, en cada una de las siguientes proposiciones que definen subprogramas:

(*a*) FUNCTION, NEXT(A, B + C, X)
(*b*) FUNCTION GRADE A, B, C
(*c*) SUBROUTINE NEW(X, Y(3), Z)
(*d*) SUBROUTINE(U, V, W)

(*a*) No debe haber coma después de FUNCTION. Además, B + C no puede ser un parámetro.
(*b*) Los parámetros A, B y C deben encerrarse entre paréntesis.
(*c*) La variable subindicada Y(3) no puede ser un parámetro en la proposición que define un subprograma.
(*d*) La subrutina no tiene nombre.

7.2 Encuentre los errores, si los hay, en cada uno de los segmentos:

(*a*)
```
FUNCTION XXX(A, N, K)
DIMENSION A(N, N + 1), B(K)
```
(*b*)
```
SUBROUTINE YYY(A, N, K)
DIMENSION A(J, K), B(N)
```

(*a*) La expresión aritmética N + 1 no puede aparecer en la proposición DIMENSION.
(*b*) La variable J no puede aparecer en la proposición DIMENSION, puesto que ella no es uno de los parámetros.

7.3 Encuentre los errores, si los hay, en cada uno de los subprogramas:

(*a*)
```
FUNCTION ADD(X, Y, Z)
Z = X + Y
RETURN
END
```
(*b*)
```
SUBROUTINE AAA(X, Y, Z)
Z = X + Y
RETURN
END
```

(*a*) Al nombre de la función ADD se le debe asignar un valor en el subprograma y no se le ha asignado.
(*b*) No hay errores. (Al nombre de un subprograma no se asigna un valor.)

7.4 Elabore una proposición que defina una función para calcular $R = \sqrt{u^2 + v^2 + w^2}$ y utilícela para evaluar

$$A = \frac{x}{\sqrt{x^2 + y^2 + z^2}}, \qquad B = \sqrt{x^4 + y^4 + z^4}, \qquad C = \sqrt{4x^2 + 9y^2 + 4z^2}$$

observemos que $x^4 = (x^2)^2$, $4x^2 = (2x)^2$, y así sucesivamente.

```
R(U, V, W) = SQRT(U*U + V*V + W*W)
A = X/R(X, Y, Z)
B = R(X*X, Y*Y, Z*Z)
C = R(2.0*X, 3.0*Y, 2.0*Z)
```

7.5 Encuentre la salida de los siguientes programas que utilizan una proposición que define funciones:

(*a*)
```
      JF(M) = M**2 - 3*M + 4
      K = 2
      L = JF(K + 2)
      M = JF(L - 3*K) + K
      WRITE(6, 10) K, L, M
10    FORMAT(1X, 3(I10, 2X))
      STOP
      END
```

(*b*)
```
      F(X, Y) = A*X + B*Y
      X = 2.0
      Y = 3.0
      A = 4.0
      B = 5.0
      C = F(6.0, 7.0)
      WRITE(6, 10) C
10    FORMAT(1X, F10.2)
      STOP
      END
```

(*a*) La primera línea define la función $JF(M) = M^2 - 3M + 4$. La segunda línea asigna 2 a K. Para las líneas 3 y 4 se tienen:

$$L \leftarrow JF(2 + 2) = JF(4) = 4^2 - 3 \cdot 4 + 4 = 8$$
$$M \leftarrow JF(8 - 3 \cdot 2) + 2 = JF(2) + 2 = (2^2 - 3 \cdot 2 + 4) + 2 = 4$$

Por lo tanto, los valores 2, 8 y 4 se imprimen en una línea, cada uno en un campo de longitud 10.

(*b*) De las líneas dos a cinco se asignan los valores 2, 3, 4 y 5 a X, Y, A y B, respectivamente. Recordamos que X y Y en F(X, Y) son variables artificiales y no tienen relación con los valores de X y Y en el programa. Específicamente, en la evaluación de C en la sexta línea, sustituimos X por 6 y Y por 7 en F(X, Y), utilizando los valores de A y B en el programa. De esta manera,

$$C \leftarrow 4 \cdot 6 + 5 \cdot 7 = 59$$

En consecuencia, el valor 59.00 se imprimirá justificado a la derecha en las primeras 10 columnas de la página de impresión.

7.6 Supongamos que se nos dan los siguientes subprogramas:

```
FUNCTION SUM(X, Y, Z)
SUM = X + Y + Z
Y = X + Y
Z = Y + Z
X = Z + X
RETURN
END
```

```
SUBROUTINE ADD(X, Y, Z, TOTAL)
Y = X + Y
Z = Y + Z
X = Z + X
TOTAL = X + Y + Z
RETURN
END
```

Encuentre la salida del segmento de programa:

```
      A = 1.0
      B = 2.0
      C = 3.0
      AMOUNT = SUM(A, B, C)
      CALL ADD(A, B, C, GT)
      WRITE(6, 10) A, B, C, AMOUNT, GT
10    FORMAT(1X, 5(F4.1, 2X))
```

Después de que se ejecute la línea 4, A = 7.0, B = 3.0, C = 6.0 y AMOUNT = 6.0. Después de que se ejecute la línea 5, A = 23.0, B = 10.0, C = 16.0 y GT = 49.0. Por lo tanto, la salida consiste en los valores 23.0, 10.0, 16.0, 6.0 y 49.0.

PROGRAMAS

7.7 Escriba un subprograma SUBROUTINE que calcule el nuevo balance mensual de una cuenta corriente, digamos en la forma

```
BANK(BAL, DEP, M, CK, N)
```

donde

BAL denota el balance mensual de la cuenta corriente,

DEP es un arreglo que lista los M depósitos para el mes,

CK es un arreglo que lista los pagos de N cheques durante el mes.

Suponemos que hay un cargo por servicio, SC, el cual es de \$1.00 por mes, de 5 centavos por cada cheque y de 2 centavos por cada depósito. Suponemos también, que DEP y CK nunca exceden de 100 elementos.

Simplemente sumamos los depósitos y restamos los cheques y el cargo por servicio. Sin embargo, debemos también considerar el caso, donde M y/o N sean iguales a cero. El subprograma es como sigue:

```
      SUBROUTINE BANK(BAL, DEP, M, CK, N)
      DIMENSION DEP(100), CK(100)
      IF(M.EQ.0) GO TO 10
      DO 100 K = 1, N
            BAL = BAL + DEP(K)
100   CONTINUE
 10   IF(N.EQ.0) GO TO 20
      DO 200 K = 1, M
            BAL = BAL - CK(K)
200   CONTINUE
 20   SC = 1.00 + M*0.02 + N*0.05
      BAL = BAL - SC
      RETURN
      END
```

7.8 Escriba un subprograma SUBROUTINE que no tenga (*a*) argumento, ni (*b*) proposición RETURN.

(*a*) El siguiente es un subprograma SUBROUTINE sin ningún argumento:

```
      SUBROUTINE NEW
      WRITE(6, 10)
10    FORMAT(1X, 'NUEVO DEPOSITARIO')
      RETURN
      END
```

La proposición

```
CALL NEW
```

en un llamado de programa producirá como efecto la impresión del mensaje

```
NUEVO DEPOSITARIO
```

(*b*) Un subprograma SUBROUTINE no necesariamente debe retornar al llamado de programa. Específicamente, si en cualquier subprograma SUBROUTINE reemplazamos la proposición RETURN por la proposición STOP, la ejecución del programa terminará por la SUBRUTINA. Por ejemplo, si la proposición RETURN se reemplaza por STOP en (*a*), el programa finaliza después de que imprima el mensaje NUEVO DEPOSITARIO.

7.9 Supongamos que A, B, C y D son arreglos lineales almacenados en la memoria con 100, 50, 75 y 200 elementos respectivamente. Supongamos que L y K también están en la memoria y $5 \leq L \leq 75$ y $L < K \leq 200$. Escriba un segmento de programa Fortran para calcular:

1. $A_{15} + A_{17} + A_{19} + \cdots + A_{77}$
2. $B_{22} + B_{23} + B_{24} + \cdots + B_{36}$
3. $C_2 + C_5 + C_8 + \cdots + C_L$
4. $D_L + D_{L+1} + D_{L+2} + \cdots + D_K$

Obsérvese que cada uno de los cálculos implica la necesidad de encontrar la suma de los términos en un arreglo lineal. En lugar de hacer cada uno de los cálculos en el programa, escribimos un subprograma function llamado SUM para calcular $X_{IN} + X_{IN+IC} + \cdots + X_{IT}$, donde X es un arreglo lineal con N elementos, IN el subíndice inicial, IC el incremento del subíndice e IT el subíndice final. El subprograma y segmento de programa Fortran son los siguientes:

Subprograma

```
      FUNCTION SUM(X, N, IN, IT, IC)
      DIMENSION X(N)
      SUM = 0.0
      DO 20 I = IN, IT, IC
            SUM = SUM + X(I)
   20 CONTINUE
      RETURN
      END
```

Segmento de programa Fortran

```
T1 = SUM(A, 100, 15, 77, 2)
T2 = SUM(B, 50, 22, 36, 1)
T3 = SUM(C, 75, 2, L, 3)
T4 = SUM(D, 200, L, K, 1)
```

7.10 Supongamos que X y Y son arreglos lineales, cada uno con N elementos. Escriba un segmento de programa para calcular

$$\frac{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \cdot \sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}}{\sqrt{x_1 y_1 + x_2 y_2 + \cdots + x_n y_n}}$$

utilizando un subprograma FUNCTION llamado INNPRO, que calcula $a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

Subprograma

```
      REAL FUNCTION INNPRO(A, B, N)
      DIMENSION A(N), B(N)
      INNPRO = 0.0
      DO 100 K = 1, N
            INNPRO = INNPRO + A(K)*B(K)
  100 CONTINUE
      RETURN
      END
```

Segmento de programa Fortran

```
UP1 = SQRT(INNPRO(X, X, N))
UP2 = SQRT(INNPRO(Y, Y, N))
DOWN = SQRT(INNPRO(X, Y, N))
ANGLE = UP1*UP2/DOWN
```

(Obsérvese la declaración de tipo necesaria para que INNPRO sea real.)

7.11 Escriba un subprograma SUBROUTINE que imprima un arreglo bidimensional fila por fila (suponiendo que el arreglo no tiene más de 13 columnas). El programa tendrá una dimensión ajustable.

```
      SUBROUTINE PRINT(A, M, N)
      DIMENSION A(M, N)
      DO 100 J = 1, M
            WRITE(6, 10) (A(J, K), K = 1, N)
   10       FORMAT(1X, 13(2X, F8.2))
  100 CONTINUE
      RETURN
      END
```

7.12 Supongamos que una matriz N×L está en la memoria. Escriba los siguientes subprogramas SUBROUTINE:

(*a*) SUBROUTINE FIND(A, N, L, K, J)

que encuentra la fila J de tal modo que A(J, K) contenga la mayor cifra en valor absoluto entre A(K, K), A(K + 1, K),..., A(N, K). (Véase problema 6.9.)

(*b*) SUBROUTINE CHANGE(A, N, L, K, J)

que intercambia los elementos de la K-ésima fila de A con los elementos correspondientes de la fila J. (Véase problema 6.9.)

(*c*) SUBROUTINE ROWMUL(A, N, L, K, J, D)

que suma D veces la K-ésima fila a la fila J. (Estos subprogramas SUBROUTINE se utilizan en la solución de ecuaciones lineales por el método de eliminación de Gauss, que se trata en el capítulo 8.)

(*a*)
```
      SUBROUTINE FIND(A, N, L, K, J)
      DIMENSION A(N, L)
      J = K
      KK = K + 1
      DO 100 I = KK, N
            IF(ABS(A(J, K)).LT.ABS(A(I, K))) J = I
  100 CONTINUE
      RETURN
      END
```

(*b*)
```
      SUBROUTINE CHANGE(A, N, L, K, J)
      DIMENSION A(N, L)
      IF(J.EQ.K) RETURN
      DO 100 I = 1, L
            T = A(K, I)
            A(K,I) = A(J,I)
            A(J,I) = T
  100 CONTINUE
      RETURN
      END
```

Obsérvese que contemplamos la posibilidad de que J = K y en este caso no se va al ciclo DO.

(*c*)
```
      SUBROUTINE ROWMUL(A, N, L, K, J, D)
      DIMENSION A(N, L)
      DO 100 I = 1, L
            A(J, I) = A(J, I) + D*A(K, I)
  100 CONTINUE
      RETURN
      END
```

Problemas suplementarios

SUBPROGRAMAS

7.13 Encuentre los errores, si los hay, en cada una de las siguientes proposiciones que definen subprogramas:

(*a*) FUNCTION, ORDER(A, B, 2*X),
(*b*) FUNCTION(I, J, K)
(*c*) SUBROUTINE SEEK(A, B, X, M (K), Z),
(*d*) SUBROUTINE LOOK(K, J(4), L, M)

7.14 Encuentre los errores, si los hay, en cada uno de los segmentos de programa Fortran:

(*a*)
```
FUNCTION BANK(A, B, N, K)
DIMENSION A(N, N – 1), C(K), B(200)
```

(*b*)
```
SUBROUTINE LOOK(X, M, N, Z)
DIMENSION X(M), Y(M, K), Z(N)
```

7.15 Encuentre los errores, si los hay, en cada uno de los subprogramas:

(*a*)
```
FUNCTION DEP(X, Y, N)
Z = X + FLOAT(N)
Y = X - N*Z
RETURN
END
```

(*b*)
```
REAL SUBROUTINE LOOK(X, Y, Z)
Z = X + Y
RETURN
END
```

7.16 Encuentre la salida del programa:

```
JG(K) = K**2 + L*K + M
K = 3
L = 4
M = 5
I = JG(6)
J = JG(L)
WRITE(6, 10) I, J
10 FORMAT(1X, 2I10)
STOP
END
```

7.17 (*a*) Escriba un subprograma FUNCTION llamado ADD(M, N) que calcule la suma de los N enteros consecutivos desde M; o sea,

$$M + (M + 1) + (M + 2) + \cdots + (M + N - 1)$$

Por ejemplo, MM = ADD(5, 4) almacenará en MM el valor 5+6+7+8 = 26

(*b*) Encuentre la salida del siguiente programa, que utiliza el anterior subprograma FUNCTION:

```
C     PROGRAMA QUE UTILIZA UN SUBPROGRAMA FUNCTION
      I = 1
      J = 2
      K = 3*I + 2*J
      I = ADD(J, K - 3)
      J = ADD(I - 1, J + 1)
      K = ADD(K, K - 3)
      WRITE(6, 10) I, J, K
10    FORMAT(1X, 3I10)
      STOP
      END
```

7.18 (*a*) Escriba un subprograma SUBROUTINE SUB(J, K, JSUM, JPROD, JDIFF) tal, que JSUM JPROD y JDIFF den la suma J+K, el producto J*K y la diferencia J – K, respectivamente.

(*b*) Encuentre la salida del siguiente programa, que utiliza el subprograma anterior.

```
C     PROGRAMA QUE UTILIZA UN SUBPROGRAMA SUBROUTINE
      J = 3
      K = 2
      CALL SUB(J, K, L, M, N)
      CALL SUB(L + M, 3*N, J, N, M)
      WRITE(6, 10) J, K, L, M, N
10    FORMAT(1X, 5(I5, 2X))
      STOP
      END
```

7.19 Escriba un subprograma FUNCTION de tipo real ADDS (J, K) que calcule la suma

$$1/J + 1/(J + 1) + 1/(J + 2) + \cdots + 1/K$$

cuando $J \leq K$ y que además tenga en cuenta el caso ADDS = 0.0 cuando $J > K$. Encuentre la salida del programa:

```
      I = 2
      J = 4
      K = 3
      X = ADDS(I, J)
      Y = ADDS(J, K)
      Z = ADDS(J, 5)
      WRITE(6, 30) X, Y, Z
30    FORMAT(1X, 3F12.2)
      STOP
      END
```

7.20 Escriba un segmento de programa para calcular:

$$A = x^2 + y^2 - 25$$
$$B = 4x^2 + 9y^2 - z^2$$
$$C = \sqrt{x^2 + y^2 - 36}$$

(Sugerencia: Utilice la proposición que define la función aritmética $F(r, s, t) = r^2 + s^2 - t^2$.)

PROGRAMAS

7.21 Dado un arreglo $A_1, A_2, \ldots, A_N$ de N elementos, la media aritmética se define como

$$(A_1 + A_2 + \cdots + A_N)/N$$

y la media geométrica se define como

$$\sqrt[N]{A_1 \cdot A_2 \cdot \ldots \cdot A_N}$$

(*a*) Escriba un subprograma SUBROUTINE para calcular la media aritmética y geométrica.
(*b*) Escriba un subprograma FUNCTION que calcule la media aritmética y geométrica.

7.22 Supongamos que C es un arreglo matricial N × N. Escriba un subprograma que encuentre el producto de la diagonal de C, cuyos elementos son C(1, 1), C(2, 2),..., C(N, N).

7.23 Supongamos que B es una matriz M × N.

(*a*) Escriba un subprograma FUNCTION para encontrar el mayor elemento en B.
(*b*) Escriba un subprograma FUNCTION que encuentre el elemento de B con mayor valor absoluto.
(*c*) Escriba un subprograma FUNCTION que encuentre la suma de los elementos de B.

7.24 En un curso que tiene exactamente 25 estudiantes se practican cuatro exámenes. Los resultados de los exámenes de cada estudiante junto con su identificación ID se perforan en una tarjeta. La calificación para cada estudiante del curso se calcula como el promedio de los tres mejores exámenes. Escriba un programa que calcule las calificaciones para los estudiantes. (Sugerencia: Una forma de calcular la suma de los tres mejores resultados es restar a la suma de los cuatro resultados, el resultado más bajo. Por lo tanto, escriba un subprograma FUNCTION para encontrar el resultado más bajo y utilícelo en la escritura del programa principal.)

7.25 Escriba un subprograma SUBROUTINE en la forma

STAT(A, N, SUM, PROM, VAR, SD)

que calcule la SUMA, la media aritmética PROM, la varianza VAR y la desviación estándar SD de los elementos de un arreglo lineal A con N elementos. (Véase problema 6.6.)

7.26 Supongamos que A es un arreglo lineal con N elementos. Escriba un subprograma SUBROUTINE:

(*a*) SORTUP(A, N) que clasifique el arreglo A, de manera que sus elementos estén en orden ascendente. (Véase problema 6.12).
(*b*) SORTDN(A, N) que clasifique el arreglo A, de manera que sus elementos estén en orden descendente.

7.27 Escriba un subprograma SUBROUTINE en la forma

DEPR(COST, N, A)

donde COST es la depreciación por el método de la suma de dígitos (véase problema 5.10) en un período de N años y A es un arreglo lineal con N elementos, tal que A(K) indica la cantidad depreciada en el K-ésimo año.

7.28 Supongamos que J y K son enteros positivos. Escriba subprogramas FUNCTION:

(*a*) LGCD(J, K) que encuentre el máximo común divisor de J y K.
(*b*) LCM(J, K) que encuentre el mínimo común múltiplo de J y K.

(Sugerencia: LCM(J, K)·LGCD(J, K) = J·K.)

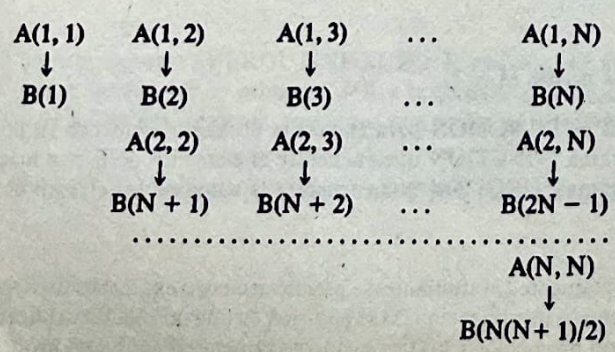
7.29 Supongamos que A es una matriz M × N. Escriba un subprograma FUNCTION:

(*a*) MAXROW(A, M, N, K) que determine la posición del máximo elemento en la K-ésima fila de A.
(*b*) MAXCOL(A, M, N, L) que encuentre la posición del máximo elemento en la L-ésima columna de A.

7.30 Consideremos que A es una matriz cuadrada N × N.

(*a*) Escriba una subrutina SYM que retorne un valor M = 1 si la matriz A es simétrica y retorne un valor M = 0 en caso contrario. (La matriz A es *simétrica* si $A_{ij}=A_{ji}$ para todo *i, j*.)
(*b*) Escriba una subrutina TRANS que lea la matriz A y retorne la traspuesta de A almacenada en A. (La *traspuesta* de A, denotada por A^T, es una matriz cuadrada B de N × N, tal que $B_{ij}=A_{ji}$ para todo *i, j*.)

7.31 Supongamos que A es una matriz N × N. Es posible almacenar los elementos de la parte triangular superior de A en un arreglo lineal B con N(N + 1)/2 elementos, como sigue:



Escriba un subprograma SUBROUTINE que lleve a cabo lo anterior. (Una matriz simétrica o una matriz triangular superior se pueden almacenar de esta manera.)

Respuestas a problemas suplementarios seleccionados

7.13 (*a*) No debe haber coma después de FUNCTION ni al final de la proposición. 2*X no puede ser un parámetro.
(*b*) El subprograma FUNCTION no tiene nombre.
(*c*) M(K) no puede ser un parámetro y no debe haber coma al final de la proposición.
(*d*) J(4) no puede ser un parámetro.

7.14 (*a*) N— no puede aparecer en la proposición DIMENSION.
(*b*) K no puede aparecer en Y(M, K), puesto que no es un parámetro.

7.15 (*a*) A DEP no se le ha asignado un valor.
(*b*) REAL no tiene sentido, puesto que a LOOK no se le ha asignado un valor.

7.16 Se imprimen los enteros 65 y 37 en campos con una longitud de 10.

7.17 (*a*) Puesto que ADD debe ser evaluada como entera, se necesita una proposición de declaración de tipo:

```
      INTEGER FUNCTION ADD(M, N)
      ADD = 0
      DO 200 K = 1, N
            ADD = ADD + (M - 1) + K
  200 CONTINUE
      RETURN
      END
```

(*b*) Se imprimen en una línea los enteros 14, 42 y 34 en campos con una longitud de 10.

7.18 (*a*) No se requiere proposición de declaración de tipo, puesto que a SUB no se le asigna ningún valor:

```
      SUBROUTINE SUB(J, K, JSUM, JPROD, JDIFF)
      JSUM = J + K
      JPROD = J*K
      JDIFF = J - K
      RETURN
      END
```

(*b*) Se imprimen en una línea los enteros 14, 2, 5, 8 y 33.

7.19 Se imprimen los números reales 1.08, 0.00 y 0.45 en campos con una longitud de 10.

7.20

```
F(R, S, T) = R*R + S*S - T*T
A = F(X, Y, 5.0)
B = F(2.0*X, 3.0*Y, Z)
C = SQRT(F(X, Y, 6.0))
```

Capítulo 8

Técnicas de programación y cálculos numéricos

8.1 INTRODUCCION

Por lo general, habrá muchas formas de escribir un programa para calcular la solución o soluciones de un problema dado. La forma que se prefiere dependerá de un gran número de factores. Primero que todo, se desea minimizar el efecto de errores de redondeo, para que los resultados sean lo más exactos posibles. (Véase sección 2.10.) En segunda instancia, se desea que el programa sea lo más eficiente posible, donde la eficiencia de un programa dependerá principalmente de dos aspectos:

1. el tiempo requerido para procesar el programa;
2. el número de localizaciones de memoria utilizadas por el programa.

En otras palabras, se trata de escribir un programa que minimice el tiempo de computación y el número de localizaciones de memoria utilizadas por el programa.

En principio, podremos escribir programas más eficientes adoptando buenos hábitos de programación. Por ejemplo, a una expresión que aparezca muchas veces se le asignará un nombre, de tal forma que se evalúe solamente una vez; una potencia tal como X**2.0 se puede escribir como X**2 ó X*X; se puede escribir

```
3.0*X**2 + 4.0*X + 5.0      en lugar de      3*X**2 + 4*X + 5
```

aunque el computador puede aceptar mezcla de modos, y así, sucesivamente.

Además de buenos hábitos de programación, con frecuencia se puede reducir considerablemente el tiempo de computación, utilizando el algoritmo adecuado. Por ejemplo, si el polinomio

$$a_1X^N + a_2X^{N-1} + \cdots + a_NX + a_{N+1}$$

se evalúa tal como se escribe, se requieren $N(N+1)/2$ multiplicaciones y N sumas. Por otra parte, usando el método de Horner (discutido en la sección 8.6), se requerirán solamente N multiplicaciones y N sumas.

Se puede reducir ampliamente la cantidad de localizaciones de memoria, si se escoge el algoritmo adecuado. En vía de ilustración, supóngase que deseamos reordenar los N elementos de un arreglo lineal N en orden inverso, es decir, de tal forma que A(1) sea el último, A(2) el siguiente al último y así sucesivamente. Una forma de hacer esto es asignar los elementos de A a otro arreglo B, (con N elementos) en la siguiente forma:

$$B(N) \leftarrow A(1), \quad B(N-1) \leftarrow A(2), \quad \ldots, \quad B(1) \leftarrow A(N)$$

lo cual se puede realizar mediante el ciclo DO:

```
      DO 99 J = 1, N
            B(N + 1 - J) = A(J)
99    CONTINUE
```

En B, los elementos están en el orden deseado. Luego, retornaremos los elementos al arreglo A asignando B(1) a A(1), B(2) a A(2) y así, sucesivamente. Obsérvese que este método requiere N localizaciones adicionales de memoria.

En otra forma, podemos realizar nuesto trabajo utilizando solamente una localización adicional de memoria. Esto es, intercambiamos A(1) y A(N), A(2) y A(N – 1) y así, sucesivamente:

```
      NN = N/2
      DO 88 J = 1, NN
            TEMP = A(J)
            A(J) = A(N + 1 – J)
            A(N + 1 – J) = TEMP
   88 CONTINUE
```

Así, este segundo método es más eficiente que el primero. (Se puede encontrar otro ejemplo en nuestra discusión en la sección 6.8, en donde se mueve el segmento de un arreglo en una posición.)

Finalmente, se debe tener en cuenta que hay un factor *humano* en la programación. Aunque escribir 2*K como K+K puede salvar una minúscula cantidad de tiempo de computador esto puede hacer el programa difícil de entender y resultará en pérdida de tiempo humano. En otras palabras, es mejor tener un procedimiento general y comprensible (independientemente de la máquina) que un programa complicado e incomprensible, lleno de trucos que dependen de la máquina.

La primera parte de este capítulo trata varias técnicas de programación para ciertos problemas que se suceden a menudo, tales como la clasificación, la búsqueda y la intercalación. La segunda parte de este capítulo está dedicada a varios cálculos numéricos, tales como la solución de ecuaciones lineales, búsqueda de raíces de polinomios y multiplicación de matrices.

8.2 CLASIFICACION

Por clasificación, queremos decir ordenamiento de elementos en alguna secuencia. Este proceso es familiar en la vida diaria, por ejemplo, los nombres de una lista telefónica están clasificados en orden alfabético. Las notas de los estudiantes se pueden clasificar de acuerdo con su número de identificación, ID y así, sucesivamente. Aunque la clasificación puede parecer un trabajo trivial, cuando es eficiente puede ser difícil tanto teórica como prácticamente.

Puesto que la clasificación y la búsqueda se usan en mantenimiento de archivos en procesamiento de datos comerciales, introduciremos primero alguna terminología. En procesamiento de datos, se clasifica un *archivo* de *registros* de acuerdo con una *clave* específica. Por ejemplo, una universidad puede mantener un *archivo* de sus estudiantes para cada estudiante, el dato que aparece en el archivo se llama *registro*. El registro puede contener varios *campos*; por ejemplo, nombre, número de seguros social, número de estudiante, sexo, clase, promedio acumulado, etc. Si deseamos clasificar el archivo de acuerdo con el orden alfabético de los nombres, entonces el campo de los nombres es la *clave*. Si fuéramos a clasificar de acuerdo con el número del seguro social, entonces éste sería la clave.

Para nuestros propósitos, consideraremos el problema de clasificar un arreglo de números A(1), A(2),..., A(N), en orden ascendente, tal que

$$A(1) < A(2) < A(3) < \cdots < A(N)$$

Supondremos que tenemos acceso directo a los elementos y que ellos están almacenados en memoria. Esto se llama una *clasificación interna.*

Algunas veces, los archivos son tan grandes que se guardan en aparatos de almacenamiento con acceso limitado. Clasificar los archivos de este tipo —llamada *clasificación externa*— es difícil y se escapa del marco de este texto.

(*a*) Clasificación por burbuja

Una forma de clasificar A es mediante el algoritmo llamado *clasificación por burbuja*, discutido en la sección 6.12. Esto es, primero comparamos A(1) y A(2) y los arreglamos en el orden deseado, en nuestro caso A(1) < A(2). Luego comparamos A(2) y A(3) y los arreglamos de tal forma que A(2) < A(3). Luego comparamos A(3) y A(4) y los arreglamos de tal forma que A(3) < A(4). Continuamos hasta comparar

A(N – 1) y A(N) y los arreglamos de tal forma que A(N – 1) < A(N). Después de este "barrido" a través de los elementos, el elemento mayor, se "empuja" al tope del arreglo, es decir, a la posición N.

Repetimos el proceso anterior para los elementos A(1), A(2),..., A(N – 1). Después de explorar todos esos elementos, se empuja el segundo mayor elemento a la posición N – 1. Continuando el proceso, después de N – 1 exploraciones, el arreglo A estará clasificado en orden creciente. El diagrama de flujo del algoritmo y su traducción Fortran aparece en el problema 6.12.

La eficiencia de este algoritmo puede demostrarse como sigue. Observemos, primero, que para cada exploración (el ciclo DO exterior), tiene lugar un intercambio solamente cuando los elementos no están arreglados en el orden deseado. En consecuencia, se puede añadir un contador para el número de intercambios en cada exploración y, si ningún intercambio tiene lugar, entonces el arreglo ha sido clasificado y el trabajo se ha finalizado.

(*b*) Clasificación selectiva

El segundo algoritmo, llamado *clasificación por intercambio* o *clasificación selectiva*, trabaja como sigue: Primero encontramos el valor más pequeño en el arreglo y lo colocamos en la primera posición; luego, encontramos el segundo valor más pequeño y lo colocamos en la segunda posición y así, sucesivamente. Establezcamos el algoritmo con más precisión:

(1) Encontrar la posición L del menor de los elementos entre A(1), A(2),..., A(N). Intercambiar A(L) con A(1).

(2) Encontrar la posición L del menor de los elementos entre A(2), A(3),..., A(N). Intercambiar A(L) con A(2).

.....

(N – 1) Encontrar la posición L del menor de los elementos entre A(N – 1), A(N). Intercambiar A(L) con A(N – 1) el arreglo A estará ahora clasificado en orden creciente.

Obsérvese que los pasos son idénticos uno a otro; luego, claramente, usaremos un ciclo DO controlado por un índice J variando de 1 a N – 1. Dentro de este ciclo DO, deseamos encontrar la posición L del menor elemento entre A(J),..., A(N), e intercambiamos entonces A(L) con A(J). El macro diagrama de flujo para el ciclo DO aparece en la figura 8-1(*a*), y el diagrama de flujo para encontrar el menor elemento entre A(J),..., A(N), aparece en la figura 8-1(*b*). Combinando estos dos diagramas en un programa, tendremos el programa de clasificación requerido:

```
C
C       PROGRAMA DE CLASIFICACION PARA A(1),..., A(N).
C
        NN = N - 1
        DO 100 J = 1, NN
C
C               ENCONTRAR LA LOCALIZACION L DEL MENOR
C
                L = J
                JJ = J + 1
                DO 200 I = JJ, N
                        IF(A(L).LT.A(I)) GO TO 200
                        L = I
  200           CONTINUE
C
C               INTERCAMBIAR A(L) CON A(J)
C
                T = A(L)
                A(L) = A(J)
                A(J) = T
  100   CONTINUE
```

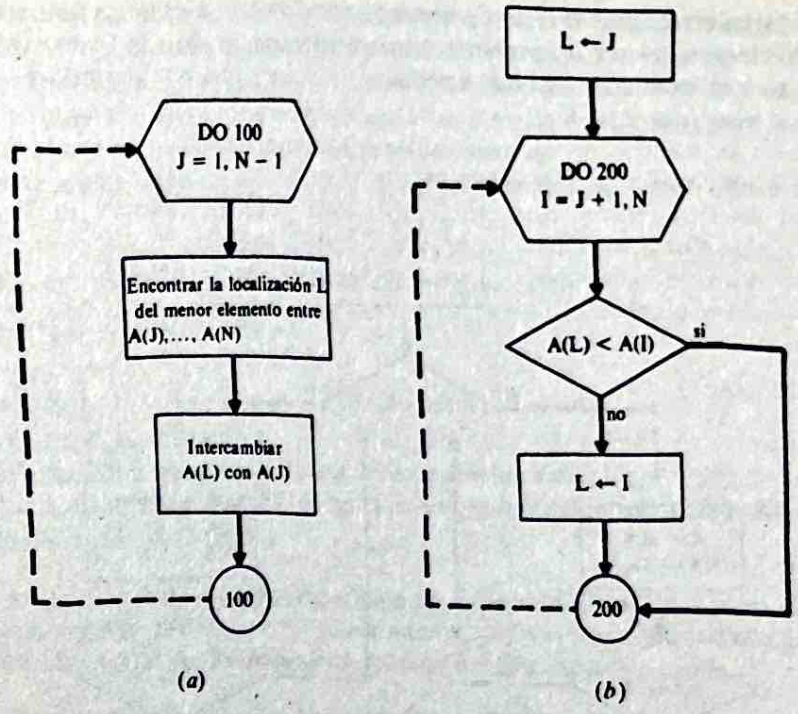


Fig. 8-1

Nota. El número de comparaciones en el método de clasificación por burbuja se puede calcular como sigue: Durante la primera exploración, hay $(N-1)$ comparaciones, durante la segunda exploración hay $(N-2)$ comparaciones y así, sucesivamente. Luego, el número total de comparaciones es

$$(N-1)+(N-2)+\cdots+2+1=\frac{N(N-1)}{2}$$

Para la clasificación selectiva, se necesitan $(N-1)$ comparaciones para encontrar el elemento menor, $(N-2)$ comparaciones para encontrar el segundo elemento menor y así, sucesivamente. Por tanto, se requieren de nuevo $N(N-1)/2$ comparaciones. Adviértase que el número promedio de intercambios será $N(N-1)/4$, la mitad del número de comparaciones. Mediante modificaciones apropiadas (véase problema 8.1), se puede reducir el número de comparaciones, pero el número promedio de intercambios será el mismo.

8.3 INTERCALACION

Supongamos que A y B son arreglos clasificados (digamos, en orden creciente) con M y N elementos, respectivamente. Deseamos intercalar A y B en un arreglo clasificado C. Una forma de hacer esto es colocar arbitrariamente A y B en C, digamos A al principio y B al final y luego clasificar C, como en la sección 8.2. Obsérvese que este método no utiliza el hecho de que A y B ya estaban clasificados. Seguidamente, damos un programa más eficiente.

Motivemos primero nuestro algoritmo. Supongamos que tenemos dos filas de estudiantes clasificados por incremento de estatura y supongamos que deseamos intercalarlos en una sola fila clasificada. Una forma de hacerlo sería ir llenando secuencialmente la nueva fila mediante la escogencia del más bajo de los dos estudiantes que estén a la cabeza de las filas, hasta que una de las filas no tenga más estudiantes. Luego los estudiantes restantes se colocan al final de la fila intercalada.

Para trasladar el algoritmo anterior a Fortran, debemos guardar siempre la pista de las posiciones de los elementos menores de A y B, que no han sido colocados en C. Denotemos por NA y NB esas posiciones. Luego, a cada paso haremos la pregunta:

Es $A(NA) < B(NB)$?

Si la respuesta es sí, colocaremos A(NA) en C y aumentaremos NA en 1; si es no, colocaremos B(NB) en C y aumentaremos NB en 1; si NA o NB alcanzan su máximo valor (esto es, M o N respectivamente), entonces colocaremos los elementos restantes del otro arreglo en C. El diagrama de flujo está en la figura 8-2.

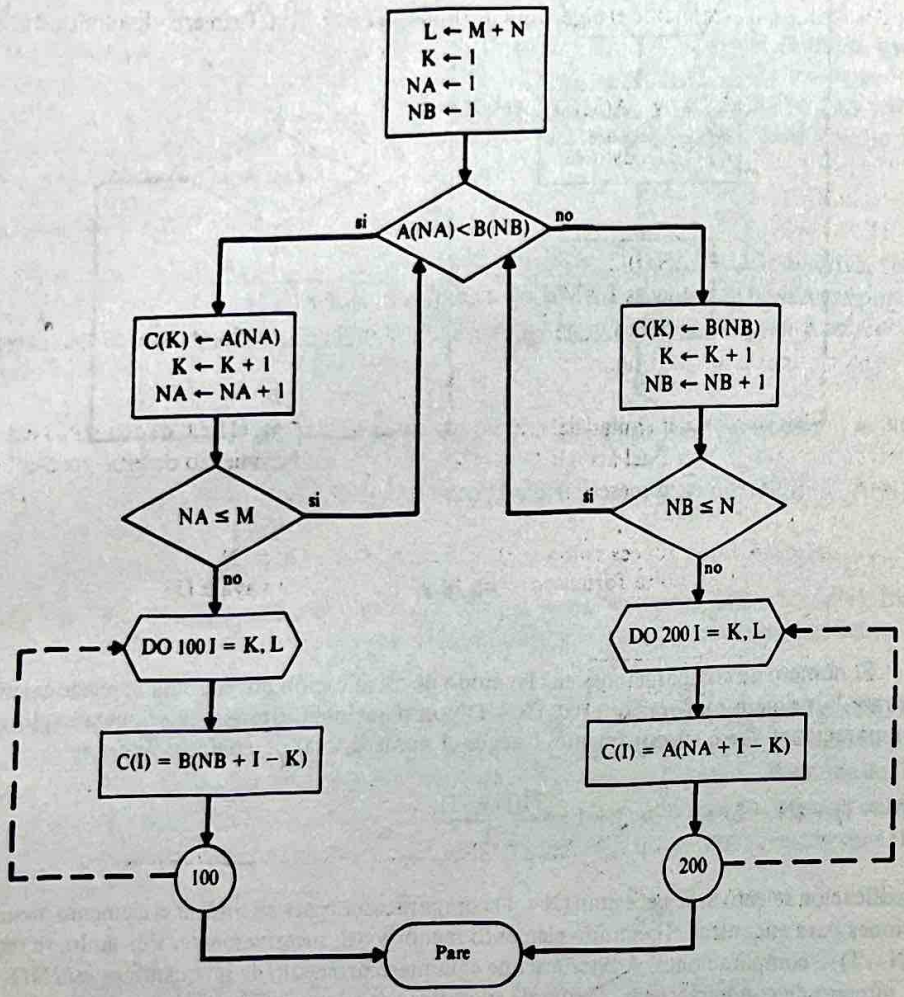


Fig. 8-2

8.4 BUSQUEDA

Supongamos que A es un arreglo lineal que está en memoria y que deseamos encontrar la posición L de un elemento D del arreglo.

(*a*) Búsqueda secuencial (Búsqueda lineal)

Puesto que no hay información adicional sobre A, se puede buscar D comparándolo uno a uno con los elementos de A. Este método se llama *búsqueda secuencial* o *búsqueda lineal* y se describe en el problema 6.11. Como se puede ver fácilmente, este método requiere N comparaciones, en el peor de los casos, para encontrar a D y N/2 comparaciones en promedio.

(*b*) Búsqueda binaria

Supongamos ahora que A es un arreglo clasificado. Puesto que el método de la búsqueda lineal de A no utiliza el hecho de que A esté clasificado, podemos usar ahora un algoritmo más eficiente, llamado *búsqueda binaria,* para encontrar D.

Motivemos primero nuestro algoritmo. Supongamos que estamos buscando un nombre en un directorio telefónico, donde los nombres estén listados alfabéticamente. Entonces, no buscamos el nombre secuencialmente desde la primera página. En lugar de eso, abrimos el directorio por la mitad y decidimos en cuál mitad del directorio está el nombre. Luego se abre el directorio en el medio de esta parte para decidir cuál parte contiene el nombre y así sucesivamente, hasta encontrar el nombre.

Describiremos ahora formalmente el algoritmo de la búsqueda binaria. Primero, dividimos el arreglo A en dos partes:

$$A(1), A(2), \ldots, A(MID) \quad y \quad A(MID), A(MID + 1), \ldots, A(N)$$

donde $MID = (1 + N)/2$. Esto es,

(1) IBEG = 1
(2) IEND = N
(3) MID = (IBEG + IEND)/2
(4) Si D = A(MID), entonces L = MID, y hemos terminado.
(5) Si D < A(MID), entonces D está en la primera mitad del arreglo; de otra forma, D está en la segunda mitad del arreglo.

Obsérvese que si D está en la primera mitad del arreglo, entonces A(MID) es el final de esta mitad del arreglo; pero si D está en la segunda mitad del arreglo, entonces A(MID) es el comienzo de esta parte del arreglo. Incorporemos esta idea en un ciclo reescribiendo el paso 5, como sigue:

(5) Si D < A(MID), entonces coloque IEND en MID y vaya a (3);
de otra forma coloque MID EN IBEG y vaya a (3).

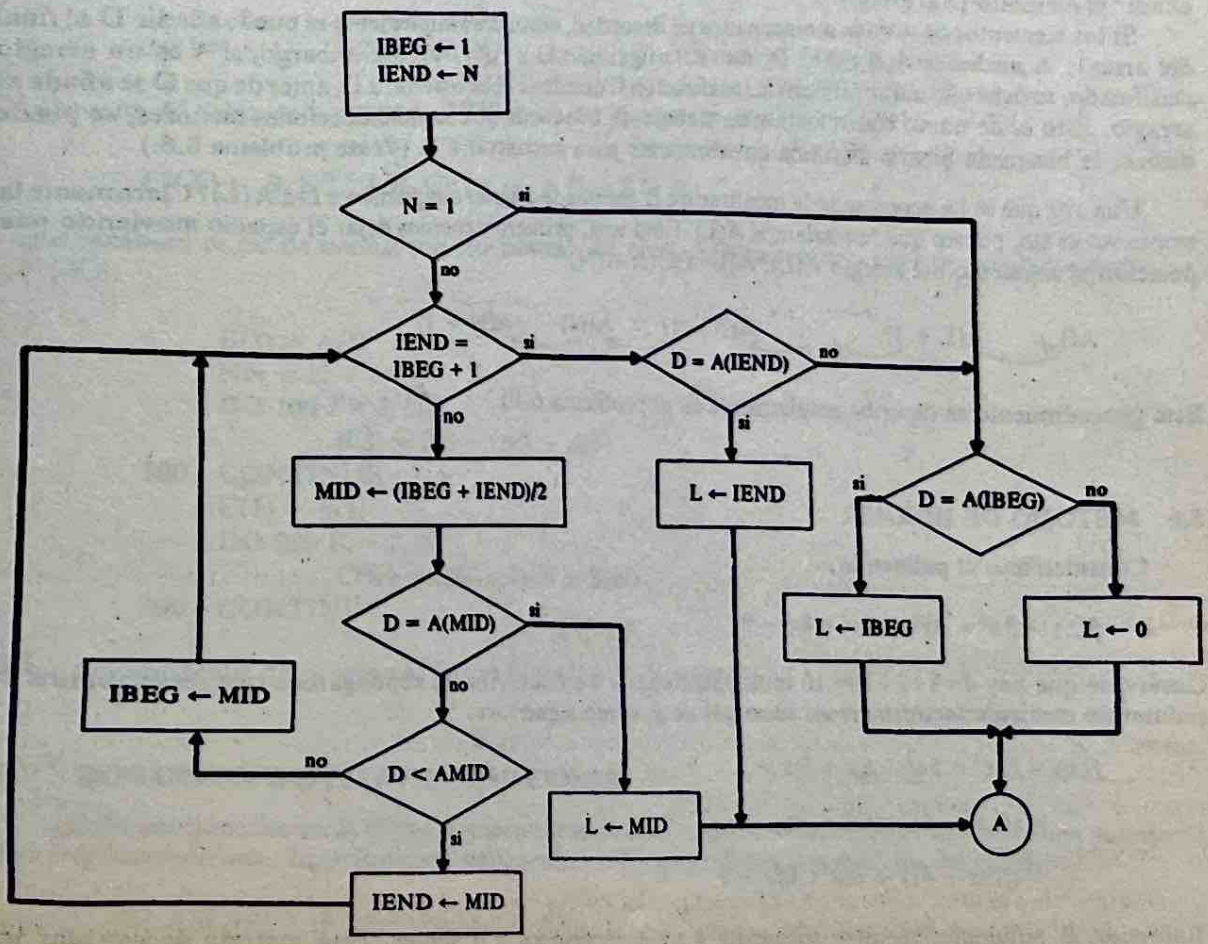


Fig. 8-3

El procedimiento anterior trabaja justo hasta que, en un momento dado, el segmento de arreglo A(IBEG),..., A(IEND) contenga solamente 2 elementos. Si el segmento tiene 2 elementos, es decir, si IEND=IBEG+1, entonces MID tendrá el valor de

(IBEG + IBEG + 1)/2

que es de nuevo IBEG, por causa de la división entera. En consecuencia, tendríamos un ciclo infinito si D > A(IBEG). Para tener en cuenta esto, chequeamos para ver si IBEG e IEND son números consecutivos.

Un último punto. Si N=1, esto es, si el arreglo originalmente empezó con un elemento, entonces ocurriría también un ciclo infinito si D no pertenece al arreglo. Luego, debemos primero probar para ver si N=1. El diagrama de flujo del algoritmo aparece en la figura 8-3. Su traducción a Fortran se deja como ejercicio al estudiante (problema 8.3).

Nota: Concluyamos esta sección comparando el método de la búsqueda binaria con el método de la búsqueda secuencial. Al utilizar el método de la búsqueda binaria, se eliminan la mitad de las alternativas cada vez que se sucede un ciclo (ver figura 8-3.) Esto es, dadas N escogencias originales, hay N/2 alternativas eliminadas al final de la primera vuelta, (N/2)/2=N/4, al final de la segunda vuelta, N/(2*2*2)=N/8, al final de la tercera vuelta y así, sucesivamente. El número total de comparaciones, en el peor de los casos, en búsqueda binaria es $\log_2 N$, lo cual es una gran ventaja sobre la búsqueda secuencial. En efecto, $\log_2 N$ es mucho menos que N/2, el número promedio de comparaciones en búsqueda secuencial. Por ejemplo, si N=10000, entonces N/2=5000, pero $\log_2 10000 \approx 14$.

8.5 ADICION DE DATOS

Supongamos que A(1), A(2),..., A(N) están en la memoria y que D no está en el arreglo A. Deseamos añadir el elemento D al arreglo.

Si los elementos de A están almacenados en desorden, entonces simplemente se puede añadir D al final del arreglo A mediante A(N+1) ← D, esto es, asignando D a A(N+1). Sin embargo, si A es un arreglo clasificado, se debe encontrar primero la localización L donde se debe insertar a D, antes de que D se añada al arreglo. Esto es de nuevo equivalente a un trabajo de búsqueda y, con modificaciones menores, se puede utilizar la búsqueda binaria discutida anteriormente para encontrar a L. (Véase problema 8.6.)

Una vez que se ha encontrado la localización L, ¿se puede asignar simplemente D a A(L)? Claramente la respuesta es no, puesto que borraríamos A(L). Entonces, primero debemos dejar el espacio moviendo una posición el segmento del arreglo A(L), A(L+1),..., A(N):

$$A(L) \quad A(L+1) \quad \cdots \quad A(N-1) \quad A(N) \quad A(N+1)$$

Este procedimiento se describe ampliamente en el problema 6.10.

8.6 METODO DE HORNER

Consideremos el polinomio

$$f(x) = 3x^4 - 5x^3 + 6x^2 + 8x - 9$$

Obsérvese que hay 4+3+2+1 = 10 multiplicaciones y 4 sumas. Ahora, supongamos que reescribimos el polinomio mediante factorizaciones sucesivas de x, como sigue:

$$\begin{aligned} f(x) &= (3x^3 - 5x^2 + 6x + 8)x - 9 \\ &= ((3x^2 - 5x + 6)x + 8)x - 9 \\ &= (((3x - 5)x + 6)x + 8)x - 9 \end{aligned}$$

Entonces, el polinomio requiere solamente 4 multiplicaciones y 4 sumas. Este método de evaluar un polinomio, llamado método de Horner, se describe a continuación, en sus líneas generales.

Consideremos ahora la ecuación polinómica general de grado N:

$$Y = F(X) = A_1X^N + A_2X^{N-1} + \cdots + A_NX + A_{N+1}$$

Evaluando el polinomio, como aparece, necesitaría

$$N + (N - 1) + (N - 2) + \cdots + 2 + 1 = N(N - 1)/2$$

N multiplicaciones y N sumas. Sin embargo, si evaluamos el polinomio usando el método de Horner:

$$Y = F(X) = ((\ldots((A_1X + A_2)X + A_3)X + \cdots)X + A_N)X + A_{N+1}$$

se requerirían solamente N multiplicaciones y N sumas. Luego este método es más eficiente.

El método de Horner tiene otra ventaja importante. Obsérvese que en cada espacio para evaluar a Y, multiplicamos por X y sumamos el coeficiente siguiente. Luego se puede usar un ciclo DO para evaluar Y, como sigue:

```
      Y = A(1)
      NN = N + 1
      DO 100 J = 2, NN
            Y = Y*X + A(J)
100   CONTINUE
```

De hecho, podemos escribir un subprograma para evaluar un polinomio de cualquier grado. (Véase problema 8.13.)

Nota: Supongamos que B(1), B(2),..., B(N+1), denote los resultados parciales en el método de Horner. Esto es,

$$B(1) = A(1), \quad B(2) = B(1)*X + A(2), \quad \ldots, \quad B(N + 1) = B(N)*X + A(N + 1)$$

Entonces, el valor de la derivada $F'(X)$ se puede mostrar, como sigue:

$$F'(X) = B_1X^{N-1} + B_2X^{N-2} + \cdots + B_{N-1}X + B_N$$

que también se puede evaluar por este método. En efecto, el siguiente segmento de programa da $F(X)$ y $F'(X)$:

```
      B(1) = A(1)
      NN = N + 1
      DO 100 J = 2, NN
            B(J) = B(J - 1)*X + A(J)
100   CONTINUE
      C(1) = B(1)
      DO 200 K = 2, N
            C(K) = C(K - 1)*X + B(K)
200   CONTINUE
```

En este segmento de programa, F(X)=B(N+1) y F'(X)=C(N).

8.7 SOLUCION DE CIERTAS ECUACIONES

Esta sección discute la forma de encontrar soluciones reales de ecuaciones que sean funciones polinomiales trigonométricas, logarítmicas o exponenciales. En general, hay dos aspectos del problema:

1. Encontrar la vecindad de una raíz.
2. Aproximar la raíz.

Por razones pedagógicas, discutimos primero los métodos de aproximación a una raíz, si se conoce la vecindad.

Considere la ecuación polinomial.

$$f(x) = x^3 - 4x^2 + 6x - 7 = 0$$

Obsérvese que $f(2) = -3$ es negativa y $f(3) = 2$ es positiva. Puesto que f es una función polinomial y su gráfico no contiene ninguna rotura en la curva, el gráfico de f cruzaría el eje x entre 2 y 3, como se muestra en la figura 8-4. El punto en el cual el gráfico cruza el eje x es una solución real de la ecuación. Damos un algoritmo, conocido como "intervalo de bisección", que encuentra una aproximación de una de tales raíces, es decir, un número que está a menos de 0.001 de la raíz exacta.

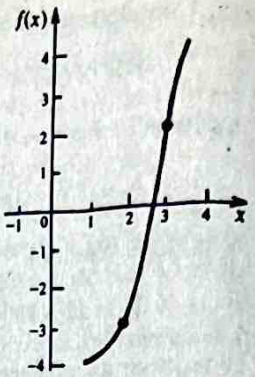


Fig. 8-4 Gráfico de $f(x)$.

(*a*) Intervalo de bisección

Primero hagamos $XN = 2.0$ y $XP = 3.0$; esto es, XN es el punto en donde f es negativo y XP es el punto en el cual f es positivo. El punto medio XM de XN y XP está dado por la fórmula

$$XM = (XN + XP)/2$$

Encontramos el punto medio y evaluamos $f(XM)$:

$$XM = (XN + XP)/2 = 2.5 \qquad \text{y} \qquad f(XM) = f(2.5) = -1.375$$

Puesto que $f(XM)$ es negativo, hacemos $XN = 2.5$. Luego encontramos XM y $f(XM)$ de nuevo:

$$XM = 2.75 \qquad \text{y} \qquad f(XM) = f(2.75) = 0.046875$$

Puesto que $f(XM)$ es positivo, hacemos $XP = 2.75$. Nótese que la raíz yace entre 2.5 y 2.75.

Continuamos el proceso anterior. Si en él, algún paso obtenemos $f(XM) = 0$; entonces XM es la raíz; de otra forma, XN y XP se van acercando más y más entre sí. Si

$$|XP - XN| < 0.001$$

la XM nueva será aproximada a la raíz verdadera alrededor de 0.001.

El diagrama de flujo para este algoritmo aparece en la figura 8-5.

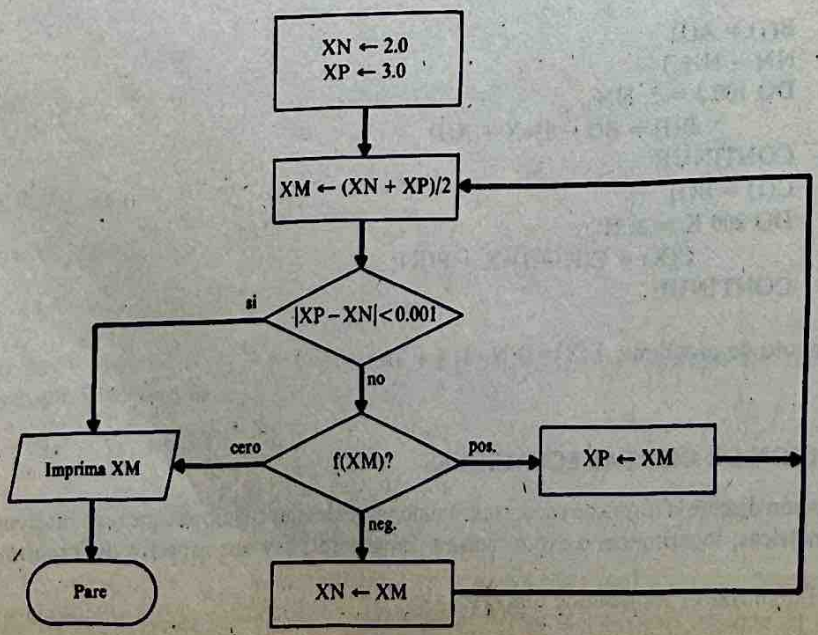


Fig. 8-5

(*b*) Localización de raíces

El algoritmo anterior trabaja si son dados la función $f(x)$ y los puntos A y B, en los cuales $f(A)$ y $f(B)$ tienen signos diferentes. La tarea más difícil es encontrar tales puntos A y B. Esto será lo que discutamos en seguida.

Supongamos que está dada una función $f(x)$ y estamos investigando una raíz entre, digamos, $x = -5$ y $x=5$. Podemos evaluar $f(x)$ para $x = -5, -4, \ldots, 5$. Si f cambia de signo para cualquier par de valores K y $K+1$, sabemos que $f(x)$ tiene una raíz entre K y $K+1$, y podemos proceder como en *(a)*. Específicamente, primero hacemos $XA = -5$ y $XB=XA+1 = -4$, y probamos el producto

```
F(XA)*F(XB)
```

Si el producto es negativo, entonces $F(XA)$ y $F(XB)$ tienen signos diferentes y podemos localizar e imprimir una raíz entre XA y XB. Por otra parte, si el producto es positivo, entonces $F(XA)$ y $F(XB)$ tienen el mismo signo y no sabemos si hay una raíz entre XA y XB. En ambos casos, hacemos que XA sea XB=-4 y luego que XB sea $XA+1$=-3 y repetimos el proceso. La repetición finaliza después que XB=5. La figura 8-6 da un diagrama de flujo para tal algoritmo. El diagrama de flujo también tiene en cuenta la posibilidad de que el producto F(XA)*F(XB) sea cero, en cuyo caso XA puede ser una raíz.

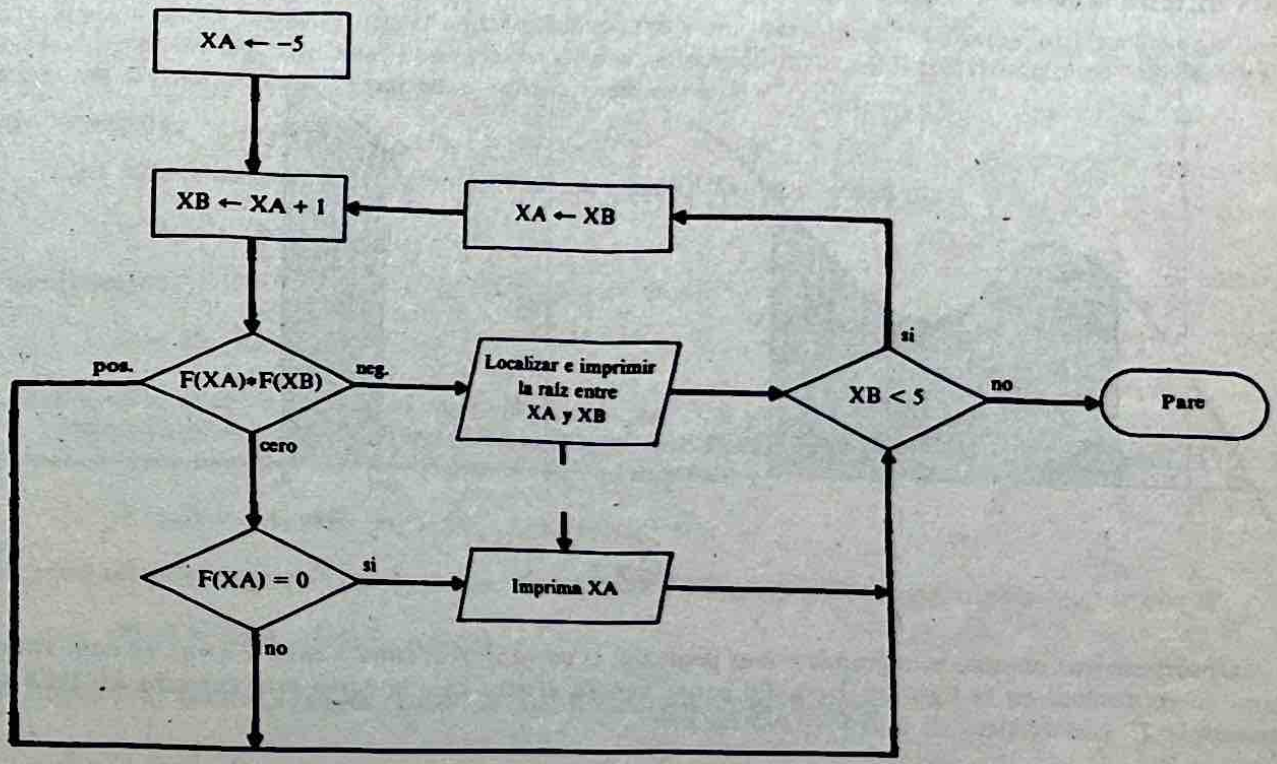


Fig. 8-6

· El hecho importante para recordar es que $f(x)$ puede tener raíces reales aun cuando no cambie de signo en los puntos. Por ejemplo, la función

$$f(x) = 8x^2 - 26x + 21$$

tiene dos raíces entre 1 y 2, aunque es positiva para todos los puntos enteros. Luego, se puede desear evaluar $f(x)$ con incrementos pequeños o con un intervalo grande antes de dar la forma de encontrar una solución. Una discusión más profunda de este problema está más allá del marco de este texto.

8.8 INTEGRACION NUMERICA

Supongamos que una función f es positiva en un intervalo $D = \{a \le x \le b\}$. Deseamos investigar el área bajo la curva $y=f(x)$ entre $x=a$ y $x=b$. Esta área está sombreada en la figura 8-7(a) y es igual a

$$\int_a^b f(x)\,dx$$

la integral de la función f de a hasta b.

Primero dividimos D en n segmentos iguales, cada uno con longitud $h=(a-b)/n$. Escogiendo los puntos

$$a_0 = a, \quad a_1 = a + h, \quad a_2 = a + 2h, \quad \ldots, \quad a_{n-1} = a + (n-1)h, \quad a_n = a + nh = b$$

Podemos afirmar que la siguiente fórmula da una aproximación para el área:

$$\text{área} \approx \tfrac{1}{2}h[f(a_0) + 2f(a_1) + 2f(a_2) + \cdots + 2f(a_{n-1}) + f(a_n)]$$

Esta fórmula se llama la "regla tropezoidal", en vista de su derivación, y se describe más abajo.

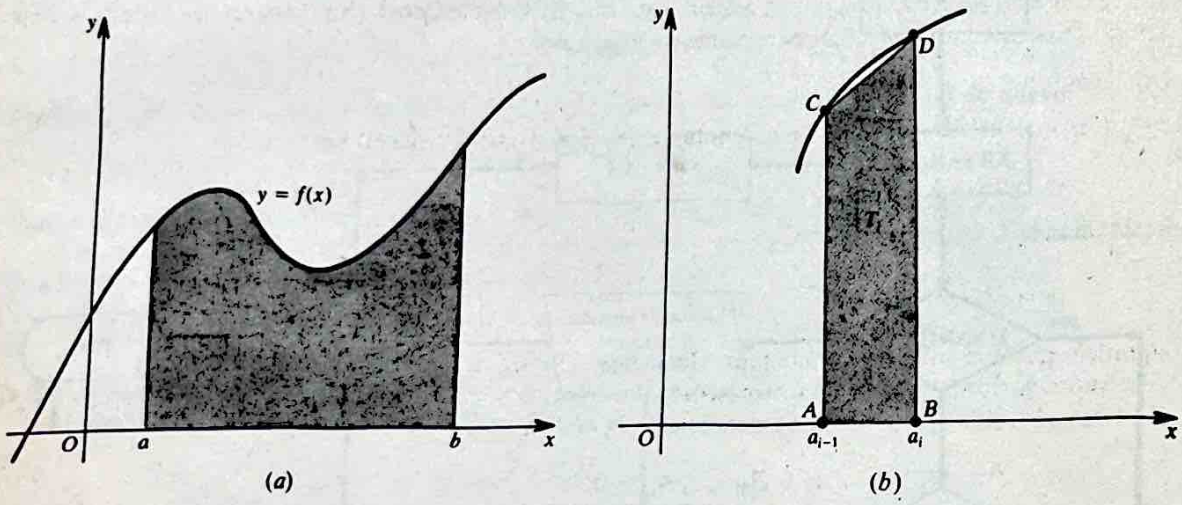


Fig. 8-7

Obsérvese que cuando n es grande y h es pequeño, la curva $y=f(x)$ entre $x=a_{i-1}$ y $x=a_i$ es casi lineal, como se representa en la figura 8-7(b). En consecuencia, el área bajo la curva está cerrada al área del tropezoide T_i con vértices A, B D y C. Nótese que

$$AC = f(a_{i-1}) \qquad \text{y} \qquad BD = f(a_i)$$

y que h es la distancia entre los lados AC y BD. Por tanto,

$$\text{área}(T_i) = \tfrac{1}{2}h[f(a_{i-1}) + f(a_i)]$$

En consecuencia, el área bajo la curva es aproximadamente igual a la suma de las áreas de los trapezoides:

$$\text{área} \approx \text{área}(T_1) + \text{área}(T_2) + \ldots + \text{área}\,(T_n)$$

lo que da la fórmula requerida.

La función $f(x)=3x^3-4x^2+6x+5$ es positiva, siempre que x sea positiva. El programa siguiente lee A, B y N, donde $0 < A < B$, calcula el área aproximada por la regla trapezoidal e imprime A, B y el área aproximada.

```
C       AREA POR LA REGLA TRAPEZOIDAL
        F(X) = ((3.0*X - 4.0)*X + 6.0)*X + 5.0
        READ(5, 10) A, B, N
   10   FORMAT(2F8.2, I5)
        H = (B - A)/FLOAT(N)
        SUM = F(A) + F(B)
        NN = N - 1
        DO 200 K = 1, NN
              SUM = SUM + 2*F(A + FLOAT(K)*H)
  200   CONTINUE
        AREA = H*SUM/2.0
        WRITE(6, 20) A, B, AREA
   20   FORMAT(1X, 2F10.2, F15.3)
        STOP
        END
```

8.9 VECTORES Y MATRICES

Esta sección investiga operaciones algebraicas entre vectores y matrices. Matemáticamente hablando, un vector es un arreglo lineal y una matriz es un arreglo bidimensional. (En el contexto de vectores y matrices, el término *escalar* se usa para números individuales).

(*a*) Vectores

Los matemáticos generalmente denotan un vector A (arreglo lineal) por:

$$A = (A_1, A_2, \ldots, A_n)$$

Por ejemplo,

$$A = (2, -3, 5, 7)$$

significa que A es un vector con cuatro elementos, $2, -3, 5$ y 7.

Supongamos que A y B son vectores, cada uno con n elementos y que X es un escalar. La suma de A y B es el vector obtenido al sumar los elementos correspondientes:

$$A + B = (A_1 + B_1, A_2 + B_2, \ldots, A_n + B_n)$$

El producto del vector A por un escalar X es el vector obtenido de multiplicar cada vector por X:

$$X \cdot A = (XA_1, XA_2, \ldots, XA_n)$$

El producto *interno* (*punto, escalar*) de A y B escrito $A \cdot B$, es el escalar

$$A \cdot B = A_1B_1 + A_2B_2 + \cdots + A_nB_n = \sum_{k=1}^{n} A_kB_k$$

La longitud (o: norma) de A, escrito $\|A\|$, es el escalar

$$\|A\| = \sqrt{A \cdot A} = \sqrt{A_1^2 + A_2^2 + \cdots + A_n^2}$$

Por ejemplo, supongamos que $A=(2, -3, 5, 7)$ y $B=(4, 2, -1, 3)$. Entonces

$$A + B = (2 + 4, -3 + 2, 5 - 1, 7 + 3) = (6, -1, 4, 10)$$
$$3A = (3 \cdot 2, 3 \cdot (-3), 3 \cdot 5, 3 \cdot 7) = (6, -9, 15, 21)$$
$$A \cdot B = 2 \cdot 4 + (-3) \cdot 2 + 5 \cdot (-1) + 7 \cdot 3 = 8 - 6 - 5 + 21 = 18$$
$$\|A\|^2 = 2^2 + (-3)^2 + 5^2 + 7^2 = 4 + 9 + 25 + 49 = 87, \text{ y así, sucesivamente } \|A\| = \sqrt{87}$$
$$\|B\| = \sqrt{16 + 4 + 1 + 9} = \sqrt{30}$$

Supongamos que A, B, C y D son arreglos lineales con N elementos y que X es un escalar. El siguiente es un segmento de programa que almacena en C la suma de A y B, almacena en D el producto escalar de X y A y almacena en DOT, ANORM Y BNORM los escalares $A \cdot B$, $\|A\|$, $\|B\|$, respectivamente.

```
      DOT = 0.0
      SUMA = 0.0
      SUMB = 0.0
      DO 100 K = 1, N
            C(K) = A(K) + B(K)
            D(K) = X*A(K)
            SUMA = SUMA + A(K)**2
            SUMB = SUMB + B(K)**2
            DOT = DOT + A(K)*B(K)
  100 CONTINUE
      ANORM = SQRT(SUMA)
      BNORM = SQRT(SUMB)
```

(*b*) Suma de matrices y multiplicación escalar

Recalcamos primero que una matriz es un arreglo bidimensional, cuyos elementos están usualmente escritos en forma rectangular, donde el primer subíndice denota la fila de la matriz y el segundo subíndice, su columna

Supongamos que $A=(a_{ij})$ y $B=(b_{ij})$ son matrices de orden $m \times n$. La suma de A y B, escrita $A + B$, es la matriz $m \times n$, cuyo elemento ij es $a_{ij}+b_{ij}$, es decir, obtenido al sumar los elementos correspondientes de A y B. El producto de la matriz A y un escalar k, escrita $k \cdot A$, es la matriz $m \times n$, cuyo elemento ij es ka_{ij}, es decir, obtenido al multiplicar cada elemento de A por k.

Por ejemplo, supongamos que A y B son las siguientes matrices de 2×3:

$$A = \begin{pmatrix} 1 & -3 & 5 \\ 2 & 0 & -6 \end{pmatrix} \quad \text{y} \quad B = \begin{pmatrix} 4 & 1 & -2 \\ 1 & -5 & -3 \end{pmatrix}$$

Entonces

$$A + B = \begin{pmatrix} 1+4 & -3+1 & 5-2 \\ 2+1 & 0-5 & -6-3 \end{pmatrix} = \begin{pmatrix} 5 & -2 & 3 \\ 3 & -5 & -9 \end{pmatrix}$$

$$-3A = \begin{pmatrix} -3\cdot 1 & -3\cdot(-3) & -3\cdot 5 \\ -3\cdot 2 & -3\cdot 0 & -3\cdot(-6) \end{pmatrix} = \begin{pmatrix} -3 & 9 & -15 \\ -6 & 0 & 18 \end{pmatrix}$$

Como en algebra ordinaria $2A - 4B = 2A + (-4B)$. Entonces,

$$2A - 4B = \begin{pmatrix} 2 & -6 & 10 \\ 4 & 0 & -12 \end{pmatrix} + \begin{pmatrix} -16 & -4 & 8 \\ -4 & 20 & 12 \end{pmatrix} = \begin{pmatrix} -14 & -10 & 18 \\ 0 & 20 & 0 \end{pmatrix}$$

Ahora supongamos que A, B, C y D son arreglos matriciales de orden M × N y que X e Y son escalares. El siguiente es un segmento de programa Fortran que almacena en C la suma de A y D y almacena en D la matriz X*A+Y*B:

```
      DO 100 L = 1, M
            DO 200 J = 1, N
                  C(I, J) = A(I, J) + B(I, J)
                  D(I, J) = X*A(I, J) + Y*B(I, J)
  200       CONTINUE
  100 CONTINUE
```

Obsérvese que, en vista de los dos subíndices, se usan ciclos DO anidados.

(c) Símbolo de sumatoria

Antes de definir multiplicación de matrices, es conveniente introducir el *símbolo de sumatoria* Σ (la letra griega sigma).

Supongamos que $f(k)$ es una expresión algebraica que involucra la variable k. Entonces la expresión

$$\sum_{k=1}^{n} f(k) \quad \text{o en forma equivalente} \quad \sum_{k=1}^{n} f(k)$$

tiene el siguiente significado: Primero hacemos $k = 1$ en $f(k)$, obteniendo

$$f(1)$$

Luego, hacemos $k=2$ en $f(k)$, obteniendo $f(2)$ y sumando esto $f(1)$, obteniendo

$$f(1) + f(2)$$

Después, hacemos $k = 3$ en $f(k)$, obteniendo $f(3)$ y añadiéndose a la suma previa, obteniendo

$$f(1) + f(2) + f(3)$$

Continuamos este proceso hasta que obtengamos la suma

$$f(1) + f(2) + f(3) + \cdots + f(n-1) + f(n)$$

(Obsérvese que a cada paso incrementamos el valor de k por 1 hasta que k sea igual a n. Notemos que se podría usar cualquier otro símbolo como i o j en lugar de k.)

Por otro lado, nuestra definición también permite que la suma sea limitada desde cualquier entero n_1 a cualquier entero n_2, donde $n_1 \leq n_2$; esto es, definimos

$$\sum_{k=n_1}^{n_2} f(k) = f(n_1) + f(n_1+1) + f(n_1+2) + \cdots + f(n_2)$$

Por ejemplo,

para $f(k) = x_k$, tenemos $\sum_{k=1}^{5} x_k = x_1 + x_2 + x_3 + x_4 + x_5$

para $f(i) = a_i b_i$, tenemos $\sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

para $f(j) = j^2$, tenemos $\sum_{j=3}^{5} j^2 = 3^2 + 4^2 + 5^2 = 9 + 16 + 25 = 50$

También tenemos:

$$\sum_{k=1}^{5} (-1)^{k+1}(1/k) = 1 - 1/2 + 1/3 - 1/4 + 1/5$$

$$\sum_{k=1}^{p} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + a_{i3} b_{3j} + \cdots + a_{ip} b_{pj}$$

$$\sum_{k=1}^{n+1} a_k x^{n+1-k} = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

(d) Multiplicación de matrices

Supongamos que $A=(a_{ij})$ es una matriz de orden $m \times p$ y que $B=(b_{ij})$ es una matriz de orden $p \times n$. El producto de A y B, escrito AB, es la matriz C de orden $m \times n$, cuyo elemento ij, c_{ij} está dado por

$$c_{ij} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{ip} b_{pj} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

[Podemos visualizar c_{ij} como el escalar obtenido por el producto interno (punto) de la i-ésima fila de A y la columna j-ésima de B.]

Por ejemplo, supongamos que A y B son las siguientes matrices de 2×3 y 3×4, respectivamente:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \text{y} \quad B = \begin{pmatrix} 5 & 6 & 7 & 8 \\ -1 & -2 & -3 & -4 \\ 0 & 9 & 0 & 9 \end{pmatrix}$$

Para obtener el producto $C=AB$, notemos que C será una matriz de 2×4. El elemento c_{11} es el producto interno de la primera fila de A y la primera columna de B:

$$c_{11} = (1, 2, 3) \cdot (5, -1, 0) = 5 - 2 + 0 = 3$$

El elemento c_{12} es el producto interno de la primera fila de A y la segunda columna de B:

$$c_{12} = (1, 2, 3) \cdot (6, -2, 9) = 6 - 4 + 27 = 29$$

De modo similar,

$$c_{13} = 7 - 6 + 0 = 1$$
$$c_{14} = 8 - 8 + 27 = 27$$

La segunda fila de C se obtiene multiplicando la segunda fila de A por cada una de las columnas de B:

$$c_{21} = 20 - 5 + 0 = 15$$
$$c_{22} = 24 - 10 + 54 = 68$$
$$c_{23} = 28 - 15 + 0 = 13$$
$$c_{24} = 32 - 20 + 54 = 66$$

En consecuencia,

$$C = \begin{pmatrix} 3 & 29 & 1 & 27 \\ 15 & 68 & 13 & 66 \end{pmatrix}$$

Obsérvese que el producto BA no está definido, puesto que el número de columnas de B (que es 4) no es igual al número de filas de A (que es 2).

Generalmente, si A es una matriz $M \times L$ y B es una matriz $L \times N$, entonces el elemento c_{ij} en el producto matricial C=AB se puede calcular por

```
      C(I, J) = 0.0
      DO 100 INDEX = 1, L
            C(I, J) = C(I, J) + A(I, INDEX)*B(INDEX, J)
  100 CONTINUE
```

En efecto, el siguiente es un subprograma completo SUBROUTINE que encuentra C:

```
      SUBROUTINE MATMUL(A, B, M, L, N, C)
      DIMENSION A(M, L), B(L, N), C(M, N)
      DO 300 I = 1, M
            DO 200 J = 1, N
                  C(I, J) = 0.0
                  DO 100 INDEX = 1, L
                        C(I, J) = C(I, J) + A(I, INDEX)*B(INDEX, J)
  100             CONTINUE
  200       CONTINUE
  300 CONTINUE
      RETURN
      END
```

8.10 ECUACIONES LINEALES

Esta sección está dedicada a la solución de un sistema de n ecuaciones lineales con n incógnitas; consideramos, primero, el caso especial de un sistema triangular y luego, el caso general utilizando la eliminación de Gauss.

(a) Sistema triangular

Se dice que el siguiente sistema de ecuaciones lineales está en forma triangular:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots\cdots\cdots + a_{1,n-1}x_{n-1} + a_{1n}x_n &= b_1 \\
a_{22}x_2 + \cdots\cdots\cdots + a_{2,n-1}x_{n-1} + a_{2n}x_n &= b_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\
a_{n-2,n-2}x_{n-2} + a_{n-2,n-1}x_{n-1} + a_{n-2,n}x_n &= b_{n-2} \\
a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\
a_{nn}x_n &= b_n
\end{aligned}$$

Aquí $x_1, x_2, \ldots, x_n$ son las n incógnitas y $a_{11}, a_{12}, \ldots, a_{1n}, \ldots, a_{nn}$ son los coeficientes. Suponemos que ninguno de los elementos de la diagonal $a_{11}, a_{22}, \ldots, a_{nn}$ son cero; por tanto, el sistema tiene una solución única. Nos referimos también a las ecuaciones por medio de $L_1, L_2, \ldots, L_n$, respectivamente.

Resolviendo para x_n en la última ecuación L_n, obtenemos

$$x_n = b_n/a_{nn}$$

Resolvamos ahora para el resto de incógnitas por el proceso conocido como *sustitución hacia atrás*. Específicamente, sustituimos x_n en la anterior ecuación a la última L_{n-1} y resolvemos para x_{n-1} obteniendo

$$x_{n-1} = (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}$$

Después, sustituimos x_n y x_{n-1} en la tercera ecuación a partir de la última, L_{n-2} y resolvemos para x_{n-2} obteniendo

$$x_{n-2} = (b_{n-2} - a_{n-2,n-1}x_{n-1} - a_{n-2,n}x_n)/a_{n-2,n-2}$$

y así, sucesivamente. Esto es, para cada $k < n$, sustituimos $x_n, \ldots, x_{k+1}$ en L_k y resolvemos para x_k obteniendo

$$x_k = (b_k - a_{k,k+1}x_{k+1} - \cdots - a_{kn}x_n)/a_{kk} = \left(b_k - \sum_{m=k+1}^{n} a_{km}x_m\right)\Big/ a_{kk}$$

(El nombre *sustitución hacia atrás* proviene del hecho de que x_k se obtiene en sentido inverso.)

Supongamos que el sistema triangular anterior está almacenado en el computador en un arreglo de N × (N+1). La subrutina BKSUB que sigue, almacenará la solución en un arreglo X:

```
          SUBROUTINE BKSUB(A, N, L, X)
          DIMENSION A(N, L), X(N)
          X(N) = A(N, N + 1)/A(N, N)
          NN = N - 1
          DO 100 J = 1, NN
                 K = N - J
C                CALCULE LA SUMA PARA X(K)
                 SUM = 0.0
                 NNN = N - K
                 DO 200 JJ = 1, NNN
                        M = K + JJ
                        SUM = SUM + A(K, M)*X(M)
    200          CONTINUE
C                RESUELVE PARA X(K)
                 X(K) = (A(K, N + 1) - SUM)/A(K, K)
    100   CONTINUE
          RETURN
          END
```

Observe los ciclos DO anidados de la subrutina. El ciclo DO exterior se refiere a la variable x_k en la ecuación L_k, y el ciclo DO interior se refiere a la sumatoria de la fórmula para x_k. Debemos, también, tener en cuenta que los x_k se obtienen en orden inverso y que la sumatoria para x_k es desde $k+1$ hasta n.

Observe también que L denota el número de columnas en A, y, por tanto, siempre será igual a N+1.

(*b*) Sistema general

Un sistema general de n sistemas lineales $E_1, E_2, \ldots, E_n$ con n incógnitas $x_1, x_2, \ldots, x_n$ puede denotarse por:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Supondremos que el sistema tiene una solución única. Nótese que siempre podemos intercambiar dos ecuaciones o sumar un múltiplo de una ecuación a otra sin cambiar la solución. Discutiremos ahora el método de eliminación de Gauss que usa estos dos pasos para transformar el sistema en un sistema equivalente, en forma triangular. Una vez el sistema esté en forma triangular, usamos la sustitución hacia atrás para encontrar su solución.

Primero eliminamos x_1 de las ecuaciones $E_2, E_3, \ldots, E_n$ por medio de los dos pasos siguientes:

1. Encontramos la ecuación E_k tal que el coeficiente a_{k1} de x_1 tenga el mayor valor absoluto entre todos los coeficientes de x_1, e intercambiamos E_k con E_1. En otras palabras, intercambiamos las ecuaciones para que a_{11} tenga el mayor valor absoluto entre los coeficientes de x_1. (Esto garantiza que $a_{11} \neq 0$ y también da una mejor precisión.)

2. Usamos la ecuación E_1, y, en particular, el coeficiente a_{11} de x_1 para eliminar los x_1 del resto de ecuaciones, como sigue: Para $k > 1$, multiplicamos E_1 por $m_{k1} = -a_{k1}/a_{11}$ y sumamos la ecuación a E_k. (El coeficiente a_{11} se llama *pivote* en este paso.)

Los pasos anteriores transforman el sistema a la forma

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

(Naturalmente, los a_{ij}, en el nuevo sistema no necesariamente serán los mismos a_{ij}, b_i del sistema original).

Repetimos el proceso anterior con el subsistema $E_2, \ldots, E_n$; esto es, eliminamos x_2 de las ecuaciones $E_3, \ldots, E_n$, mediante los dos pasos análogos:

1. Intercambiamos las ecuaciones (excluyendo E_1) para que el elemento pivote a_{22} tenga el mayor valor absoluto entre los coeficientes de x_2 del subsistema.

2. Para $k > 2$, multiplicamos E_2 por $m_{k2} = -a_{k2}/a_{22}$ y la sumamos a E_k.

Esos pasos transforman el sistema a la forma

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + \cdots + a_{1n}x_n &= b_1 \\ a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + \cdots + a_{2n}x_n &= b_2 \\ a_{33}x_3 + a_{34}x_4 + \cdots + a_{3n}x_n &= b_3 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n3}x_3 + a_{n4}x_4 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Después de repetir el proceso $n-1$ veces, el sistema será transformado a una forma triangular la cual puede resolverse por sustitución hacia atrás.

Deseamos ahora trasladar el algoritmo anterior en un subprograma SUBROUTINE llamado GAUSS. Vamos a suponer que el sistema de ecuaciones lineales está almacenado en memoria por medio de su *matriz aumentada*, es decir, el arreglo N × (N+1)

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} & A_{1,N+1} \\ A_{21} & A_{22} & \dots & A_{2N} & A_{2,N+1} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & \dots & A_{NN} & A_{N,N+1} \end{pmatrix}$$

donde la columna N+1 de la matriz es la columna de constantes de las ecuaciones. Nuestra subrutina GAUSS utilizará las tres subrutinas siguientes, que están dadas en el problema 7.14:

1. SUBROUTINE FIND(A, N, L, K, J)

   la cual encuentra la fila J, de modo que A(J, K) contiene el mayor valor absoluto entre A(K, K), A(K+1, K),..., A(N, K).

2. SUBROUTINE CHANGE(A, N, L, K, J)

   la cual intercambia los elementos de la fila K de A con los elementos correspondientes de la fila J.

3. SUBROUTINE ROWMUL(A, N, L, K, J, D)

   la cual suma D veces la fila K a la fila J.

(En esas subrutinas, A es un arreglo matricial de N × L.)

Podemos eliminar x_1 de las ecuaciones $E_2, \dots, E_n$ utilizando el segmento del programa

```
      CALL FIND(A, N, N + 1, 1, J)
      CALL CHANGE(A, N, N + 1, 1, J)
      DO 100 I = 2, N
            CALL ROWMUL(A, N, N + 1, 1, I, −A(I, 1)/A(1, 1))
100   CONTINUE
```

Análogamente, podemos eliminar x_2 de las ecuaciones $E_3, \dots, E_n$ con el segmento del programa.

```
      CALL FIND(A, N, N + 1, 2, J)
      CALL CHANGE(A, N, N + 1, 2, J)
      DO 200 I = 3, N
            CALL ROWMUL(A, N, N + 1, 2, I, −A(I, 2)/A(2, 2))
200   CONTINUE
```

Y así, sucesivamente. El subprograma completo utiliza un ciclo DO para cada uno de los pasos anteriores y luego usa la subrutina BKSUB para encontrar la solución después de que el sistema esté en forma triangular.

```
      SUBROUTINE GAUSS(A, N, L, X)
C
C     SOLUCION POR ELIMINACION DE GAUSS
C
      DIMENSION A(N, L), X(N)
      NN = N − 1
      DO 99 K = 1, NN
            CALL FIND(A, N, N + 1, K, J)
            CALL CHANGE(A, N, N + 1, K, J)
            KK = K + 1
            DO 88 I = KK, N
                  CALL ROWMUL(A, N, N + 1, K, I, −A(I, K)/A(K, K))
88          CONTINUE
99    CONTINUE
      CALL BKSUB(A, N, N + 1, X)
      RETURN
      END
```

Problemas

TECNICAS DE PROGRAMACION

8.1 Modificar el programa de clasificación por burbuja del problema 6.12 para incluir un contador que cuente el número de intercambios. Si en alguna exploración ningún cambio tiene lugar, terminar el programa.

8.2 Trasladar a Fortran el diagrama de flujo de la figura 8-2 que intercala dos arreglos clasificados A y B en un arreglo clasificado C. Pruebe el programa con los datos:

A: 1, 5, 6, 12, 14, 21; B: 2, 3, 9, 16, 18, 19, 24, 28

8.3 Trasladar a Fortran el diagrama de flujo de la figura 8-3 que da el algoritmo de búsqueda binaria.

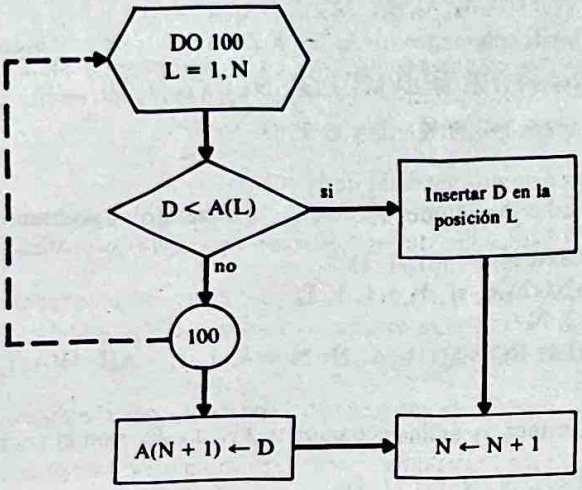


Fig. 8-8

8.4 (*Adición de datos*) Supongamos que A(1), A(2),..., A(N) están en orden creciente y supongamos que D no está en el arreglo A. El diagrama de la figura 8-8 adiciona el arreglo A, utilizando una búsqueda lineal para encontrar la posición L donde se debe insertar a D. Trasladar el diagrama de flujo a Fortran.

8.5 (*Adición de datos*) Modificar el programa del problema 8.4 de tal forma que se use una búsqueda binaria para encontrar la posición L donde se debe insertar D.

8.6 (*Búsqueda y adición de datos*) Supongamos que A(1), A(2),..., A(N) están en orden creciente y que D es un artículo dado. Si D pertenece al arreglo A, deseamos encontrar su posición, pero si D no pertenece a A, entonces deseamos añadirlo a A, encontrando la posición L donde se debe insertar.

(*a*) Escribir un programa utilizando búsqueda secuencial.
(*b*) Escribir un programa utilizando búsqueda binaria,

8.7 (*Clasificación por inserción*) Se describe en seguida otro algoritmo de clasificación llamado clasificación por inserción. La idea principal es insertar un artículo en la posición apropiada relativa a una porción clasificada del archivo. Es decir, clasificamos los dos primeros artículos A(1) y A(2) y luego insertamos el tercer artículo, de tal forma que A(1), A(2) y A(3) queden ordenados apropiadamente. En general, A(K) se inserta en el segmento del arreglo clasificado A(1),..., A(K – 1) de tal modo que A(1),..., A(K) quede clasificado apropiadamente.

Escribir un programa para clasificación por inserción. Obsérvese que la posición en la cual deberá colocarse a A(K) puede encontrarse por búsqueda secuencial o por búsqueda binaria. Discutir el número máximo de comparaciones que se pueden requerir en cada caso. Discutir las desventajas de la clasificación por inserción.

8.8 (*Intercalación-Clasificación*) Otro algoritmo de clasificación bastante eficiente y conocido es el algoritmo llamado *intercalación-clasificación*. Seguidamente damos el algoritmo. En la primera exploración, se clasifica el primer par, el segundo par, el tercer par y así sucesivamente, hasta que todos los pares queden clasificados, es decir, de tal forma que cada

$$\{A_1, A_2\}, \{A_3, A_4\}, \{A_5, A_6\}, \{A_7, A_8\}, \{A_9, A_{10}\}, \ldots$$

sea clasificado. En la segunda exploración, se intercala el primer par de los segmentoos del arreglo clasificado de dos elementos; luego, se intercala el segundo par de los segmentos de los arreglos clasificados de dos elementos y así, sucesivamente. Al final de la segunda exploración está clasificada cada una de las cuádruplas

$$\{A_1, A_2, A_3, A_4\}, \{A_5, A_6, A_7, A_8\}, \{A_9, A_{10}, A_{11}, A_{12}\}, \ldots$$

Los pares de cuádruplas son entonces intercaladas en la tercera exploración y así sucesivamente, hasta que el arreglo entero quede clasificado. Escribir un programa para implementar esta idea. Demostrar que este algoritmo requiere aproximadamente $n \log_2 n$ comparaciones.

8.9 (*a*) Escribir una subrutina DELETE que borre un segmento de arreglo dado A(IN), A(IN+1),..., A(LAST) de un arreglo A. [Sugerencia: Mover un número apropiado de localizaciones en el segmento de arreglo A(LAST+1),..., A(N).]

(*b*) Supongamos que el arreglo ID(1), ID(2),..., ID(N) está en memoria, pero algunos de sus elementos son cero. Escribir un programa para comprimir el arreglo ID, utilizando la subrutina DELETE.

8.10 Supongamos que cada vendedor en una tienda de departamentos tiene asignado un número de identificación ID. Por cada venta hecha, se perfora una tarjeta con el ID del vendedor y el monto de la venta. Al finalizar el día, se recogen esas tarjetas, pero en cualquier orden. Escribir un programa para calcular las ventas totales hechas por cada vendedor.

8.11 Supongamos que el Centro de Actividades de Computación cobra a sus usuarios a una tasa de $600.00 por hora. Cada usuario tiene asignado un número de identificación ID. Por cada trabajo procesado por el computador, se perfora una tarjeta con el número ID del usuario y el tiempo en segundos que requirió el trabajo. Estas tarjetas no están clasificadas. Suponiendo que se cobra un sobreprecio de $5.00 por cada trabajo procesado, escribir un programa para calcular la cuenta total que debe pagar cada usuario. Utilizar una tarjeta de delimitación para terminar el programa.

8.12 En el archivo maestro de la Compañía Telefónica Adler, cada registro consiste de un número de cuenta del cliente y su costo básico mensual. Estos registros están perforados en tarjetas, un registro por tarjeta y el archivo maestro está clasificado de acuerdo con el número de cuenta. Cuando se hace una llamada telefónica de larga distancia, se perfora una tarjeta con el número de cuenta del cliente y el costo de la llamada de larga distancia. Estas tarjetas no están clasificadas en ninguna forma.

Supongamos que las cuentas telefónicas son procesadas al finalizar cada mes. Suponiendo que la Compañía Telefónica Adler no tiene sino cien clientes y que las tarjetas de larga distancia están separadas del archivo maestro por una tarjeta en blanco y que el paquete de datos termina con una tarjeta delimitadora, escriba un programa para calcular las cuentas de cada cliente.

CALCULOS NUMERICOS

8.13 Escribir un subprograma función HORNER(A, N, X) que calcule el valor del polinomio de grado $N-1$

$$A_1X^{N-1} + A_2X^{N-2} + \cdots + A_{N-1}X + A_N$$

8.14 Encontrar una raíz de $f(x)=x^3-4x^2+6x+7$ entre $x=2$ y $x=3$ trasladando a Fortran el diagrama de flujo de intervalo por bisección de la figura 8-5.

8.15 Localizar y encontrar las tres raíces reales de $f(x)=x^3+x^2-37x+33$ entre $x=-10$ y $x=10$, utilizando el algoritmo descrito en la figura 8-6 para localizar las raíces y el algoritmo de intervalo por bisección para encontrar las raíces.

8.16 Encontrar dos raíces reales de $f(x)=x^2-\text{sen } x$.

8.17 Encontrar el área bajo la curva $y=x^2-3x+4$ entre $x=3$ y $x=5$.

8.18 Encontrar el área bajo la curva $y=x^2+\text{sen } x$ entre $x=0$ y $x=2$.

8.19 Escribir un subprograma subrutina NEWTON(A, N, X, Y, Z), que calcule $Y=F(X)$ y $Z=F'(X)$ donde

$$F(X) = A_1X^{N-1} + A_2X^{N-2} + \cdots + A_{N-1}X + A_N$$

y $F'(X)$ es la derivada de $F(X)$. Compare con el problema 8.13. (Sugerencia: Véase la discusión al final de la sección 8.6.)

8.20 (*Newton-Raphson*) Considere la ecuación algebráica

$$F(X) = 0$$

Sea X_0 un valor en la vecindad de una solución de la ecuación. Formamos la secuencia

$$X_1 = X_0 - F(X_0)/F'(X_0), \quad X_2 = X_1 - F(X_1)/F'(X_1), \quad \ldots$$

esto es,

$$X_{N+1} = X_N - F(X_N)/F'(X_N)$$

[Aquí $F'(X)$ denota la derivada de $F(X)$.] Bajo ciertas condiciones, cuya discusión se escapa del marco de este texto, la secuencia $X_0, X_1, X_2, \ldots$ converge a una raíz de $F(X)$. (Este proceso se llama el método de Newton-Raphson.) Escribir un programa que utilice este método para calcular $\sqrt{5}$, esto es para encontrar una solución de

$$X^2 - 5 = 0$$

Escoger $X_0=3$ y terminar el programa cuando $|X_{N+1} - X_N| \le 0.001$.

8.21 Utilizar el método de Newton-Raphson para encontrar las raíces con tres cifras decimales de la ecuación $x^2+x-11=0$. Primero escoger $x_0=-5$ y luego, $x_0=5$.

VECTORES, MATRICES Y ECUACIONES LINEALES

8.22 Supongamos que A = (2, −1, 0, −3), B = (1, −4, −4, 3), C = (1, 3, −2, 5).

(*a*) Encontrar: (*i*) 2A − 3B; (*ii*) 5A − 3B − 4C; (*iii*) A·B, A·C, y B·C; (*iv*) ‖A‖, ‖B‖, y ‖C‖.
(*b*) Escribir un programa Fortran para calcular lo anterior.

8.23 Supongamos que

$$A=\begin{pmatrix}1 & -1 & 2\\ 0 & 3 & 4\end{pmatrix},\quad B=\begin{pmatrix}4 & 0 & -3\\ -1 & -2 & 3\end{pmatrix},\quad C=\begin{pmatrix}2 & -3 & 0 & 1\\ 5 & -1 & -4 & 2\\ -1 & 0 & 0 & 3\end{pmatrix},$$

y $D=\begin{pmatrix}2 & 2\\ 3 & -1\end{pmatrix}$

Encontrar: (*a*) A + B (*d*) AC (*g*) D^3
(*b*) 3A − 4B (*e*) BC (*h*) DA
(*c*) 2A + 3B (*f*) D^2 (*i*) DB

8.24 Supongamos que seis tarjetas contienen, cada una, tres números reales. Escribir un programa que lea esos números por filas de dos arreglos matriciales A y B, y que calcule e imprima: (*a*) 5A + 2B, (*b*) 3A − 7B, (*c*) AB, (*d*) BA, (*e*) A^2.

8.25 Considere la matriz A de 6 × 6 con 0 en la diagonal principal, 1 sobre la diagonal principal y − 1, bajo la diagonal principal:

$$A=\begin{pmatrix}0 & 1 & 1 & 1 & 1 & 1\\ -1 & 0 & 1 & 1 & 1 & 1\\ -1 & -1 & 0 & 1 & 1 & 1\\ -1 & -1 & -1 & 0 & 1 & 1\\ -1 & -1 & -1 & -1 & 0 & 1\\ -1 & -1 & -1 & -1 & -1 & 0\end{pmatrix}$$

Escribir un programa que almacene A en el computador y calcule e imprima A, A^2 y A^3.

8.26 Encontrar la solución del sistema:

$$\begin{aligned}3x+4y-5z+2s+6t &= 10\\ 5y+7z-4s-6t &= -1\\ 2z+6s-3t &= -35\\ 8s+3t &= -11\\ 5t &= 35\end{aligned}$$

8.27 Encontrar la solución del sistema:

$$\begin{aligned}2x+3y+4z-5s+7t &= -35\\ 8x-2y-3z+9s+3t &= 53\\ 4y+6z-3s-2t &= -33\\ 5x-7y+8z+3s-9t &= -19\\ 3x+5y-2z+4s+6t &= 27\end{aligned}$$

8.28 (*Gauss-Jordan*) Supongamos que un sistema de N ecuaciones con N incógnitas, está almacenado en el computador por medio de un arreglo matricial A de N × (N+1). Se puede modificar el método de Gauss y transformar la matriz a la forma:

$$\begin{pmatrix}A_{11} & 0 & \dots & 0 & A_{1,N+1}\\ 0 & A_{22} & \dots & 0 & A_{2,N+1}\\ \dots & \dots & \dots & \dots & \dots\\ 0 & 0 & \dots & A_{NN} & A_{N,N+1}\end{pmatrix}$$

en lugar de la forma triangular, la solución del sistema es entonces:

$$X_1 = A_{1,N+1}/A_{11},\quad X_2 = A_{2,N+1}/A_{22},\quad \dots,\quad X_N = A_{N,N+1}/A_{NN}$$

(Este proceso se llama método de Gauss-Jordan.) Escriba un programa para implementar este algoritmo y probar el programa con el sistema del problema 8-24.

8.29 Sea A una matriz cuadrada de orden N × N. Una matriz B de orden N × N se llama *inversa* de A si AB=BA=I, donde I es la matriz identidad, la matriz con 1 en la diagonal principal y 0 en el resto. La inversa de A, si existe, es única y se denota por A^{-1}. Un algoritmo para encontrar A^{-1} es como sigue: Primero formamos la matriz N × 2N

$$(A, I) = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} & 1 & 0 & \dots & 0 \\ A_{21} & A_{22} & \dots & A_{2N} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & \dots & A_{NN} & 0 & 0 & \dots & 1 \end{pmatrix}$$

es decir, la mitad izquierda de la matriz es A y la mitad derecha es I. Utilizando una modificación del método de Gauss, se transforma la matriz anterior en una matriz de la forma

$$(I, B) = \begin{pmatrix} 1 & 0 & \dots & 0 & B_{11} & B_{12} & \dots & B_{1N} \\ 0 & 1 & \dots & 0 & B_{21} & B_{22} & \dots & B_{2N} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & B_{N1} & B_{N2} & \dots & B_{NN} \end{pmatrix}$$

es decir, donde la mitad izquierda es ahora I. La mitad derecha, B, es entonces la inversa de A. Escriba un programa para implementar este algoritmo.

Respuestas a problemas seleccionados

8.22 (*i*) (1, 10, 12, −15) (*ii*) (3, −5, 20, −44) (*iii*) −3, −16, 12 (*iv*) $\sqrt{14}, \sqrt{42}$, y $\sqrt{39}$

8.23 (*a*) $\begin{pmatrix} 5 & -1 & -1 \\ -1 & 1 & 7 \end{pmatrix}$ (*d*) $\begin{pmatrix} -5 & -2 & 4 & 5 \\ 11 & -3 & -12 & 18 \end{pmatrix}$ (*g*) $\begin{pmatrix} 26 & 18 \\ 27 & -1 \end{pmatrix}$

(*b*) $\begin{pmatrix} -13 & -3 & 18 \\ 4 & 17 & 0 \end{pmatrix}$ (*e*) $\begin{pmatrix} 11 & -12 & 0 & -5 \\ -15 & 5 & 8 & 4 \end{pmatrix}$ (*h*) $\begin{pmatrix} 2 & 4 & 12 \\ 3 & -6 & 2 \end{pmatrix}$

(*c*) $\begin{pmatrix} 14 & -2 & -5 \\ -3 & 0 & 17 \end{pmatrix}$ (*f*) $\begin{pmatrix} 10 & 2 \\ 3 & 7 \end{pmatrix}$ (*i*) $\begin{pmatrix} 6 & -4 & 0 \\ 13 & 2 & -12 \end{pmatrix}$

8.26 $x = 3, y = -2, z = 5, s = -4, t = 7$

8.27 $x = 2, y = 1, z = -5, s = 3, t = -1$

Capítulo 9

Información en forma de caracteres. Variables lógicas y operaciones

9.1 INTRODUCCION

El Fortran se desarrolló principalmente para ser utilizado en aplicaciones que tuviesen que ver con el manejo de fórmulas (FORmula TRANslation). Sin embargo, también se puede utilizar en la manipulación de información en forma de caracteres. Recordamos que hemos sido capaces de imprimir letreros colocando el mensaje entre un par de apóstrofes en la proposición FORMAT. Por ejemplo,

```
      WRITE(6, 10)
10    FORMAT(1X, 'EL FINAL')
```

imprimirá el mensaje

```
 EL FINAL
```

este mensaje tiene ocho caracteres, E, L, un espacio en blanco (que denotaremos más adelante por ␣), F, I, N, A y L. Fortran también nos permite leer, almacenar y manipular tal cadena de caracteres, por ejemplo, para listar nombres por orden alfabético. En este capítulo, discutiremos la manipulación de símbolos empleada por Fortran.

Fortran puede utilizarse también, tanto para ejecutar operaciones lógicas como para evaluar expresiones del mismo tipo. Esto también se discutirá en este capítulo.

9.2 ALMACENAMIENTO DE CARACTERES

Cuando los caracteres se almacenan en la memoria del computador, el almacenamiento se hace por medio de códigos. En una máquina binaria, los caracteres se almacenan como una secuencia de 0's y 1's. Claramente, esta secuencia se puede interpretar como un número binario. Este número determina el equivalente de la información en forma de caracteres. Es necesario hacer hincapié en un punto. La representación interna de

```
1234
```

almacenada como una cadena de caracteres es completamente diferente de la representación interna de 1234, almacenada como un entero.

Hay un límite en el número máximo de caracteres que se pueden almacenar en una localización de memoria. Este número, que varía de máquina a máquina, se llamará la *capacidad de caracteres* (o simplemente *capacidad*) de la máquina y la designaremos por la letra M. Para máquinas de las series IBM 360/370, M=4 y de las series CDC 6000, M=10. (Para hacer programas manejables, se aconseja suponer M=4.)

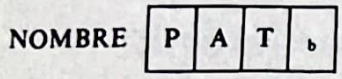
Teóricamente, se puede utilizar cualquier localización real o entera para almacenar información en forma de caracteres. Sin embargo, cuando se manipula información en forma de caracteres es más seguro utilizar localizaciones enteras, por cuanto ellas evitan cualquier error de redondeo que pueda presentarse.

Una manera de almacenar información en forma de caracteres es por medio de la proposición no ejecutable DATA. (La proposición DATA se discute con detalle en la sección 10.2.) Por ejemplo, la proposición

```
      DATA NOMBRE/'PAT'/
```

colocada en el programa antes de cualquier proposición ejecutable, almacenará la cadena de caracteres PAT en una localización llamada NOMBRE. Los apóstrofes le indican al computador que PAT se almacenará como caracteres. Las diagonales se utilizan para encerrar el detalle de entrada, separándolo del nombre de la variable.

Suponiendo que la capacidad de caracteres M=4, PAT se almacenará en la memoria como sigue:

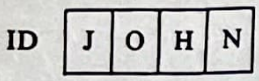


es decir, la cadena de caracteres se almacenará en la localización de memoria justificada a la izquierda y los espacios restantes a la derecha se rellenan con blancos.

Por otra parte, supongamos que se da la proposición

```
DATA ID/'JOHNSON'/
```

es decir, la cadena de caracteres JOHNSON se debe almacenar en la localización de memoria ID. De nuevo, la cadena de caracteres se almacenará en la memoria justificada a la izquierda. Sin embargo, suponiendo que M=4 es la capacidad, los caracteres en exceso a la derecha se truncan y solamente JOHN será almacenado:



Es decir que solamente se almacenan en la memoria los M = 4 caracteres más hacia la izquierda.

Los siguientes comentarios deben tenerse en cuenta:

1. Algunos compiladores no permiten que las constantes de caracteres aparezcan en proposiciones ejecutables. En este caso, se puede almacenar información en forma de caracteres a partir de una proposición de asignación. Por ejemplo, la proposición

   ```
   NOMBRE = 'PAT'
   ```

   tendrá el mismo efecto que

   ```
   DATA NOMBRE/'PAT'/
   ```

   Es decir, la cadena de caracteres colocada entre apóstrofes se almacenará en una localización de memoria llamada NOMBRE.

2. Con M=4, se requerirían al menos dos localizaciones para almacenar la cadena JOHNSON. Por ejemplo, con cualquiera de las siguientes proposiciones,

   ```
   DATA IDA/'JOHN'/
   DATA IDB/'SON'/
   ```

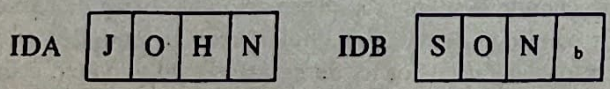
   ó

   ```
   DATA IDA/'JOHN'/, IDB/'SON'/
   ```

   ó

   ```
   DATA IDA, IDB/'JOHN', 'SON'/
   ```

   se almacenará JOHN en IDA y SON b en IDB:

3. Se puede también utilizar un campo H, en lugar de los apóstrofes en la proposición DATA, por ejemplo,

   ```
   DATA NOMBRE/3HPAT/
   ```

   también almacena la cadena de caracteres PAT en NOMBRE. (Véase sección 9.5.)

4. Aunque algunos compiladores no permiten que las constantes de caracteres aparezcan en proposiciones ejecutables como en (1), siempre se puede copiar el contenido de una localización en otra, por medio de una proposición de asignación. Por ejemplo, suponiendo que M=4, el segmento de programa

```
DATA NOMBRE/'PAT'/
NEXT = NOMBRE
```

almacenará PAT_b, el contenido de NOMBRE, en una localización de memoria, cuyo nombre es NEXT:

Obsérvese que tanto NEXT como NAME son variables enteras. Es aconsejable no hacer mezclas de tipos en tales proposiciones de asignación.

Advertencia. Cuando la longitud *l* de una constante de caracteres en una proposición DATA exceda la capacidad de caracteres M, algunos compiladores pueden dar un mensaje de error. Existen otras posibilidades que se discutirán en el problema 10.3. Sin embargo, todos los compiladores dan el mismo resultado cuando $l \leq M$.

9.3 CAMPO A

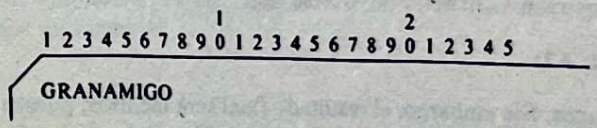
Otra manera de almacenar la información en forma de caracteres es por medio de lectura, es decir, a partir de la proposición READ. Cuando se lee o imprime información en forma de caracteres se debe utilizar el campo A, cuya especificación tiene la forma

Aw

donde *w* es el tamaño del campo. (Se puede también utilizar *rAw*, donde *r* es el factor de repetición.)

(*a*) Entrada

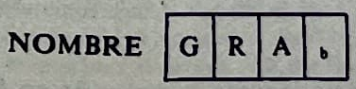
Supongamos que la capacidad M=4 y que se ha perforado una tarjeta de datos como la siguiente:



Además, supongamos que la instrucción de entrada es

```
      READ(5, 10) NOMBRE
10    FORMAT(A3)
```

Esta le dice al computador que la cadena de caracteres en las columnas 1-3 se almacenará en la localización llamada NOMBRE. En este caso, el tamaño del campo *w*=3 no excede la capacidad de caracteres M; de esta manera la cadena de caracteres se almacenará en la memoria justificada a la izquierda con blancos a la derecha:



Por otra parte, supongamos que la instrucción de entrada fuera

```
      READ(5, 20) NOMBRE
20    FORMAT(A10)
```

Aquí el tamaño del campo *w*=10 excede a M. En este caso, solamente los M=4 caracteres más a la derecha del campo se almacenarán en la memoria:

NOMBRE	O	b	b	b

Obsérvese que la proposición DATA

```
DATA NOMBRE/'GRANAMIGO'/
```

tendría un resultado diferente. Es decir, la proposición DATA llevará a la memoria solamente los M=4 caracteres más a la izquierda:

NOMBRE	G	R	A	N

Los resultados anteriores se pueden resumir como sigue:

Reglas sobre entrada de caracteres utilizando el campo A. Supongamos que M es la capacidad de caracteres y w es el tamaño del campo dado en la proposición FORMAT.

1. Si $w \leq$ M, entonces los w caracteres del campo se almacenarán en la memoria justificados a la izquierda con M $-$ w blancos a la derecha.
2. Si $w >$ M, entonces solamente se almacenarán en la memoria los M caracteres más a la derecha.

Por consiguiente, para leer el nombre GRANAMIGO utilizando la anterior tarjeta de datos con M=4, requerimos al menos tres localizaciones de memoria. Por ejemplo,

```
      READ(5, 10) L1, L2, L3
10    FORMAT(3A4)
```

almacenará los primeros cuatro caracteres GRAN en L1, los segundos cuatro caracteres AMIG en L2 y los terceros cuatro caracteres O$_{bbb}$ en L3:

L1	G	R	A	N

L2	A	M	I	G

L3	O	b	b	b

En caso de que la proposición FORMAT se cambie por

```
10    FORMAT(2A4, A2)
```

entonces sólo se leen 9 columnas. Sin embargo, el resultado final será idéntico; puesto que el carácter O se almacenará en L3 justificado a la izquierda con tres blancos a la derecha.

Obviamente, se puede también utilizar un arreglo para almacenar el nombre GRANAMIGO, como sigue:

```
      DIMENSION L(3)
      READ(5, 10) L
10    FORMAT(3A4)
```

Finalmente, hacemos hincapié en que los blancos se tienen en cuenta como caracteres blancos cuando en la lectura se utilizan campos A, aunque ellos se toman como ceros cuando en la lectura se usan campos numéricos.

(*b*) Salida

Primero, recordamos que si una localización de almacenamiento contiene información en forma de caracteres, entonces debe contener exactamente M caracteres (es decir, se completará con blancos a la izquierda, si la cadena de caracteres original tiene menos de M caracteres). Antes de establecer formalmente las reglas sobre el uso del campo A a la salida, damos un ejemplo.

Supongamos que la capacidad M=4 y que NOMBRE contiene

NOMBRE	P	A	T	b

Entonces

```
      WRITE(6, 20) NOMBRE
20    FORMAT(1X, A4)
```

imprimirá el contenido de NOMBRE

PAT_b

En este caso, el tamaño del campo dado w es igual al de la capacidad M=4. Sin embargo, si la proposición FORMAT es

```
20    FORMAT(1X, A2)
```

donde el tamaño del campo dado w=2 es menor que M=4, entonces de los caracteres almacenados en la memoria, solamente se imprimen los dos que están más a la izquierda

PA

De otro lado, si la proposición FORMAT es

```
20    FORMAT(1X, A7)
```

donde el tamaño del campo dado w=7 es mayor que M=4, entonces los M=4 caracteres almacenados en la memoria se imprimirán justificados a la derecha del campo, como sigue:

$_{bbb}PAT_b$

En otras palabras, la salida consiste en $w-M=3$ blancos seguidos del contenido de NOMBRE.

Los resultados anteriores pueden resumirse como sigue:

Reglas sobre salida de caracteres utilizando el campo A. Supongamos que M es la capacidad de caracteres y w es el tamaño del campo dado en la proposición FORMAT.

1. Si $w \leq M$, entonces de los caracteres almacenados en memoria, la salida resultante será los w caracteres más a la izquierda; es decir, se truncarán los $M-w$ caracteres más a la derecha.
2. Si $w > M$, entonces la salida resultante será $w-M$ blancos seguidos de los M caracteres almacenados en memoria.

EJEMPLO 9.1

Queremos perforar tarjetas de datos y almacenar e imprimir 200 nombres tal que la salida aparezca como:

```
BROWN          ERIC
EVERLING       MICHAEL
LIPSON         AUDREY
........,      .......
```

Suponiendo que ningún nombre (primero o último) exceda 15 caracteres, perforamos el último nombre justificado a la izquierda en las columnas 1-15 y el primer nombre justificado a la izquierda en las columnas 17-31. (En esta forma, al menos un blanco separa al último nombre del primero.)

Si suponemos que la capacidad M=4, cada nombre requiere al menos de 8 localizaciones de memoria para almacenar los posibles 31 caracteres. Por lo tanto, usamos un arreglo bidimensional llamado NOMBRE de 200×8. Para almacenar e imprimir el arreglo, fila por fila, debemos emplear ciclos DO implícitos. El segmento de programa es el siguiente:

```
      DIMENSION NOMBRE(200, 8)
      DO 99 K = 1, 200
            READ(5, 15) (NOMBRE(K, I), I = 1, 8)
15          FORMAT(8A4)
            WRITE(6, 25) (NOMBRE(K, I), I = 1, 8)
25          FORMAT(1X, 8A4)
99    CONTINUE
```

9.4 MANEJO DE LA INFORMACION EN FORMA DE CARACTERES

Como se anotó en la sección 9.2, el contenido de cualquier localización de memoria es una secuencia de 0's y 1's (suponiendo que tenemos una máquina binaria). Por lo tanto, se pueden ejecutar operaciones de relación entre los equivalentes numéricos de las cadenas de caracteres. Esto es, la información en forma de caracteres se puede manejar, como veremos en la siguiente discusión.

(*a*) Clasificación

Frecuentemente se desea clasificar datos por sus nombres en caracteres, es decir, listar nombres por orden alfabético. Para ordenar alfabéticamente cadenas de caracteres, se debe conocer cómo se codifican internamente los caracteres alfabéticos. Afortunadamente, la mayoría de los compiladores hacen la codificación de los caracteres alfabéticos en orden ascendente. Esto es, el equivalente numérico del carácter A es menor que el de B y así, sucesivamente. Por ejemplo, supongamos que IA e IB contienen la siguiente información en forma de caracteres:

IA | A | L | A | N | IB | D | A | L | E |

Entonces, el equivalente numérico de ALAN almacenado en IA es menor que el equivalente numérico de DALE almacenado en IB. Por consiguiente, la expresión de relación

IA.LT.IB

es verdadera. Por esta razón, los varios métodos que existen para clasificación de valores numéricos, también se pueden usar para clasificar cadenas de caracteres. El problema 9.8 utiliza el método de clasificación por burbuja para clasificar una lista de nombres.

(*b*) Análisis de texto

Supongamos que se perfora una frase S en una tarjeta de datos. (Recordamos que una tarjeta de datos tiene 80 columnas.) Consideremos el problema de escribir un segmento de programa que

1. Imprima la frase S.
2. Cuente el número de veces que la letra E aparece en S.

Si deseáramos imprimir solamente S, podríamos almacenar cuatro caracteres por cada localización, mediante un arreglo lineal de 20 elementos, como sigue:

```
      DIMENSION L(20)
      READ(5, 15) L
15    FORMAT(20A4)
      WRITE(6, 25) L
25    FORMAT(1X, 20A4)
```

Sin embargo, con tal programa no podríamos contar el número de veces que aparece la letra E. A fin de lograr esto, debemos almacenar un carácter por cada localización por medio de un arreglo lineal con 80 elementos. El programa es el siguiente:

```
      DIMENSION L(80)
      DATA JJ/'E'/
      READ(5, 15) L
15    FORMAT(80A1)
      WRITE(6, 25) L
25    FORMAT(1X, 80A1)
      N = 0
      DO 99 K = 1, 80
            IF(L(K).EQ.JJ) N = N + 1
99    CONTINUE
      WRITE(6, 35) N
35    FORMAT(1X, 'LA LETRA E APARECE', 1X, I2, 1X, 'VECES')
```

Obsérvese que utilizamos una proposición DATA para almacenar la letra E en una localización llamada JJ y luego un ciclo DO para comparar JJ con cada letra de la frase S.

9.5 CAMPO H

Recalcamos que en la sección 3.8 imprimimos mensajes en nuestra salida en la proposición FORMAT, encerrándolos entre apóstrofes. Por ejemplo, supongamos que X contiene el valor 123.45. Entonces, la pareja WRITE-FORMAT

```
      WRITE(6, 10) X
10    FORMAT(1X, 'EL BALANCE ES $', F8.2)
```

producirá

```
EL␣ BALANCE ␣ES␣$␣␣123.45
```

Análogamente,

```
      WRITE(6, 20)
20    FORMAT('1', 'CUENTA NUMERO', 10X, 'BALANCE')
```

imprimirá el siguiente encabezamiento al comienzo de una nueva página:

```
CUENTA NUMERO␣␣␣␣␣␣␣␣␣␣ BALANCE
```

Se puede imprimir mensajes utilizando el campo H (campo Hollerith). La especificación del campo tiene la forma

wH seguido de los *w* caracteres que deben imprimirse

En particular, las anteriores proposiciones FORMAT pudieron perforarse, equivalentemente, como sigue:

```
10    FORMAT(1X, 15 HEL BALANCE ES $, F8.2)
20    FORMAT(1H1, 13 HCUENTA NUMERO, 10X, 7HBALANCE)
```

Hay que recalcar que los *w* caracteres no se encierran entre apóstrofes (a menos que deban imprimirse). Las siguientes especificaciones dan una lista de campo H y sus equivalentes especificaciones de campo literal:

Campo H	Campo Literal
9HSECUENCIA	'SECUENCIA'
8HEL FINAL	'EL FINAL'
18HAHORA ES EL TIEMPO	'AHORA ES EL TIEMPO'

La desventaja de la especificación del campo H es que los espacios deben contarse con exactitud. Observamos que los blancos dentro del campo H no se ignoran, puesto que un blanco es un carácter.

El campo H también puede utilizarse en la entrada. En particular, se puede usar el campo H para especificar una cadena de caracteres en una proposición DATA. Por ejemplo,

```
DATA I/8HMAURICIO/, J/5HDIANA/
```

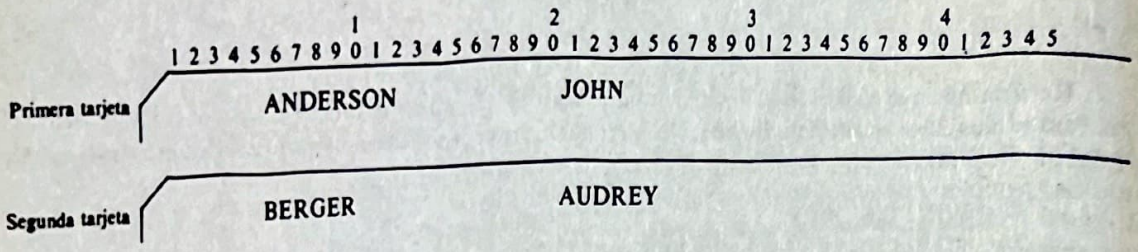
tiene el mismo efecto que

```
DATA I/'MAURICIO'/, J/'DIANA'/
```

Se puede utilizar el campo H en una proposición FORMAT de entrada. En tal caso, los caracteres ficticios (o: espacios en blanco) deben aparecer en la proposición FORMAT. Ilustramos este caso con el siguiente ejemplo.

EJEMPLO 9.2

Supongamos que las dos primeras tarjetas de datos se perforan como sigue:



y que se ejecuta el siguiente segmento de programa:

```
      DO 100 K = 1, 2
            READ(5, 10)
10          FORMAT(5X, 15HAAAAAABBBBBCCCCC, 15HXXXXXYYYYYZZZZZ)
10          WRITE(6, 10)
100   CONTINUE
```

Cuando se ejecuta la proposición READ, las cadenas de caracteres de las columnas 6-20 y de las columnas 21-35 se sustituyen por los caracteres en las especificaciones del Campo H en la proposición FORMAT, es decir, la proposición FORMAT efectivamente llega a ser

```
10  FORMAT(5X, 15HANDERSON          , 15HJOHN          )
```

En consecuencia, cuando se ejecuta la proposición WRITE, se imprime el nombre que aparece en la tarjeta de datos. Análogamente, cuando la proposición READ se ejecuta por segunda vez, el nombre en la segunda tarjeta se lee en la proposición FORMAT y se imprime cuando se ejecuta la proposición WRITE. Por consiguiente, la salida aparecerá en la forma siguiente:

```
ANDERSON      JOHN
BERGER        AUDREY
```

9.6 CONSTANTES Y VARIABLES LOGICAS

Mientras en el capítulo 4 se discutía la proposición IF lógica, al mismo tiempo presentábamos la forma más simple de una expresión lógica, la expresión de relación. Estas son expresiones que relacionan dos expresiones aritméticas por uno cualquiera de los siguientes operadores de relación:

.EQ. .NE. .LT. .LE. .GT. .GE.

Las expresiones de relación también se denominan *expresiones lógicas* porque cada una de ellas asume un valor lógico que puede ser VERDADERO o FALSO. En Fortran, estas dos constantes lógicas se representan por

.TRUE. y .FALSE.

respectivamente. Insistimos en que las palabras TRUE y FALSE deben estar precedidas y seguidas por puntos.

Las localizaciones de almacenamiento (variables) que se utilizan para almacenar constantes lógicas se llaman *variables lógicas*. Como todos los nombres de variables, un nombre de variable lógica consta de uno a seis caracteres alfanuméricos, con la condición de que el primero sea alfabético. Los nombres de variables lógicas deben declararse por medio de la proposición de especificación de tipo LOGICAL. Por ejemplo,

```
LOGICAL I, A45X, CAR
```

declara que I, A45X y CAR son variables lógicas. Los arreglos lógicos también se pueden declarar por la proposición LOGICAL. Por ejemplo,

```
LOGICAL A(15)
```

no solamente declara que A es un arreglo lógico, sino que también le asigna 15 localizaciones de memoria. Una proposición LOGICAL es no ejecutable, y también debe aparecer en el programa antes de cualquier proposición ejecutable.

9.7 OPERADORES Y EXPRESIONES LOGICOS

Las expresiones lógicas son más complejas que las expresiones de relación y pueden construirse a partir de los siguientes tres operadores lógicos Fortran (conectores):

.AND.
.OR.
.NOT.

De nuevo insistimos en que las palabras AND, OR y NOT deben estar precedidas y seguidas por puntos.

Dada cualquiera de dos expresiones lógicas, *expl* 1 y *expl* 2, podemos formar las expresiones lógicas compuestas:

expl 1.AND.*expl* 2
expl 1.OR.*expl* 2
.NOT.*expl* 1

Los valores lógicos de estas expresiones compuestas dependen de los valores lógicos de *expl* 1 y *expl* 2 y se muestran en las tablas 9-1 y 9-2 (donde T significa verdadero y F significa falso). Nótese que

1. El operador .AND. es verdadero solamente cuando *expl* 1 y *expl* 2 son *ambos verdaderos.*
2. El operador .OR. es falso solamente cuando *expl* 1 y *expl* 2 son *ambos falsos.*
3. El operador .NOT. es un *operador unario* que cambia el valor lógico de cualquier expresión lógica de T a F y de F a T.

Estos operadores son lo mismo que los conectores utilizados en español para la formación de frases compuestas.

Tabla 9-1

expl 1	*expl* 2	*expl* 1.AND.*expl* 2	*expl* 1.OR.*expl* 2
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Tabla 9-2

expl 1	.NOT.*expl* 1
T	F
F	T

EJEMPLO 9.3

(*a*) Considere las siguientes expresiones lógicas:

(*i*) (I.LT.J).OR.(K.GT.10)
(*ii*) (X.GT.1.0).AND.(K.LT.(I + 10))
(*iii*) .NOT.(X.GT.0.0)

Tenemos

(*i*) es verdadera si I < J o si K > 10.

(*ii*) es verdadera si X > I y K < I+10.

(*iii*) es verdadera si "X > 0" es falsa, esto es si X ≤ 0.

(*b*) Un error común que cometen los estudiantes principiantes es traducir $1 \leq x \leq 2$ a Fortran cuando escriben

```
1.0.LE.X.LE.2.0
```

Se debe entender que $1 \leq x \leq 2$ significa que $1 \leq x$ y que $x \leq 2$. Por lo tanto, el equivalente correcto en Fortran es

```
(1.0.LE.X).AND.(X.LE.2.0)
```

Aunque las expresiones lógicas se utilizan frecuentemente en proposiciones IF lógica, los problemas de álgebra booleana y de lógica proposicional pueden programarse y resolverse utilizando estas expresiones.

9.8 ASIGNACION DE VALORES LOGICOS

Se puede almacenar una constante lógica en una variable lógica por medio de una proposición de asignación. Por ejemplo,

```
LOGICAL A
A = .TRUE.
```

asigna a la constante .TRUE. la localización llamada A. En general, la proposición de asignación lógica tiene la forma

Variable lógica = expresión lógica

En consecuencia,

```
LOGICAL B, C
B = X.LT.Y
C = B
```

asigna a B el valor .TRUE. o .FALSE., según que X < Y sea verdadero o falso; este valor de B se asigna luego a C. Asimismo,

```
LOGICAL D
D = (R.GT.S).OR.(X.LE.Y)
```

asigna a D el valor .TRUE. si R > S es verdadero o si X ≤ Y es verdadero (o si ambos son verdaderos); de lo contrario se asigna a D el valor .FALSE. (Nótese que los paréntesis pueden utilizarse en la forma convencional.)

Una proposición DATA también se puede utilizar para asignar valores lógicos. Por ejemplo,

```
LOGICAL A, B
DATA A/.TRUE./, B/.FALSE./
```

asigna el valor .TRUE. a la localización A y .FALSE. a B durante la compilación. (La proposición DATA se discute con detalle en la sección 10.2.)

El valor asociado con el nombre de un subprograma FUNCTION puede ser lógico. El siguiente subprograma FUNCTION retorna el valor .TRUE. solamente si X es un elemento del arreglo A.

```
      LOGICAL FUNCTION CHECK(A, N, X)
      DIMENSION A(N)
      CHECK = .TRUE.
      DO 99 I = 1, N
            IF(X.EQ.A(I)) RETURN
   99 CONTINUE
      CHECK = .FALSE.
      RETURN
      END
```

Obsérvese que la declaración de tipo en la proposición que define la función, le indica al computador que CHECK es una variable lógica y no una variable real.

9.9 CAMPO L

Las variables lógicas se pueden almacenar o imprimir por medio de proposiciones READ y WRITE. Esto se realiza por la vía del campo L, cuya especificación de formato tiene la forma

Lw

donde *w* denota el tamaño del campo. (Se puede también utilizar *rLw*, donde *r* es el factor de repetición.) El uso del campo L se da como sigue.

(*a*) Entrada

Supongamos que el código de formato *Lw* corresponde a la variable lógica A. Entonces a A se asigna el valor .TRUE., si el carácter principal diferente de blanco en el campo especificado es la letra T. Si el carácter principal diferente de blanco es la letra F o todo el campo consiste en blancos, entonces se asigna .FALSE. a A. (Algunos compiladores Fortran consideran cualquier otro símbolo principal distinto de T o F como .FALSE., mientras que otros compiladores darán un mensaje de error.)

(*b*) Salida

En la salida, se imprime la letra T o F justificada a la derecha en su campo, según que la correspondiente variable lógica tenga el valor .TRUE. o .FALSE. Por ejemplo, supongamos que A y B son variables lógicas que contienen los valores .TRUE. y .FALSE., respectivamente; entonces, la instrucción de salida

```
      WRITE(6, 10) A, B
   10 FORMAT(1X, L5, 2X, L3)
```

imprimirá T en la columna 5 y F en la columna 10. Por lo tanto, el código de formato *Lw* colocará $w-1$ blancos antes de imprimir la letra T o F.

EJEMPLO 9.4

(*a*) Supongamos que una tarjeta de datos se perfora como sigue:

```
                    1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
T H E   F I R S T   D A Y
```

Además, supongamos que la instrucción de entrada es:

```
      LOGICAL A, B, C
      READ(5, 10) A, B, C
   10 FORMAT(L3, 2L5)
```

Los tamaños de campo son 3, 5 y 5, respectivamente. El primer carácter diferente de blanco en las columnas 1-3 es T, en las columnas 4-8 es F y en las columnas 9-13 es T. Por lo tanto, .TRUE. se asigna a las variables lógicas A y C y .FALSE. se asigna a B.

(*b*) Supongamos que la proposición FORMAT en la parte (*a*) se cambia por

```
10  FORMAT(3L5)
```

En este caso, T, I y D son los primeros caracteres diferentes de blanco en los tres campos. Por lo tanto, algunos compiladores asignarán .TRUE. a A y .FALSE. a B y C, pero otros compiladores darán un mensaje de error.

(*c*) Encuentre la salida del siguiente segmento de programa:

```
      LOGICAL X, Y, Z
      J = 5
      X = .FALSE.
      Y = 3*J.GT.12
      Z = Y.AND.X
      WRITE(6, 20) X, Y, Z
20    FORMAT(1X, 3L10)
```

En la línea tres se asigna .FALSE. a X. Puesto que 3*J=5 > 12 es verdadero, .TRUE. se asigna a Y por medio de la línea cuatro. Aunque Y es verdadero, X es falso; por lo tanto se le asigna .FALSE. a Z en la línea cinco. Por consiguiente, se imprime F, T y F en las columnas 10, 20 y 30, respectivamente.

9.10 JERARQUIA DE LAS OPERACIONES

Como los operadores lógicos incluyen expresiones de relación, las que a su turno involucran expresiones aritméticas y operadores aritméticos, es importante comprender la jerarquía completa de las operaciones. El orden en el que se llevan a cabo las operaciones se indica enseguida

1. Operaciones aritméticas
   (*a*) Exponenciación (**)
   (*b*) Multiplicación y división (* y /)
   (*c*) Suma y resta (+y−)
2. Operadores de relación
3. Operadores lógicos
   (*a*) .NOT.
   (*b*) .AND.
   (*c*) .OR.

(Como se anotaba en la sección 2.6, trataremos el más y el menos al mismo nivel como suma y resta binaria.)

Notamos que las operaciones aritméticas se ejecutan primero. Las operaciones de relación entonces se llevan a cabo antes que las operaciones lógicas. No hay prioridad entre las seis operaciones de relación. O sea que los operadores de relación se ejecutan en el orden en que aparecen de izquierda a derecha. Como en las expresiones aritméticas, es posible cambiar esta jerarquía con el uso de paréntesis. Más aún, es sumamente aconsejable utilizar paréntesis cada vez que su uso mejore la comprensión del orden de ejecución de las operaciones. Por ejemplo, es aconsejable escribir

```
(M.GE.1).OR.(X.NE.Y)      y      (K.GT.10).OR.((X.GE.0.2).AND.(Y.EQ.3.0))
```

en lugar de

```
M.GE.1.OR.X.NE.Y      y      K.GT.10.OR.X.GE.0.2.AND.Y.EQ.3.0
```

aunque ellas sean equivalentes.

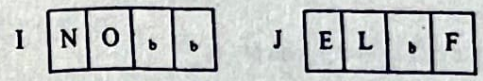
Problemas resueltos

DATOS EN FORMA DE CARACTERES

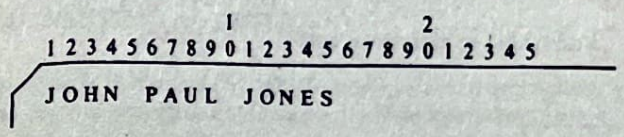
9.1 Suponiendo que la capacidad de caracteres es M=4, encuentre el dato asignado a I y a J en la siguiente proposición

```
DATA I/'NO'/, J/'EL FINAL'/
```

La cadena NO se almacena en I justificada a la izquierda con 4 – 2=2 blancos a la derecha, pero en J solamente se almacenan los M=4 caracteres más hacia la izquierda (incluyendo los blancos) de la cadena EL FINAL:



9.2 Supongamos que M=4 y que una tarjeta de datos se perfora como sigue:



Encuentre los datos almacenados, si la instrucción de entrada es

(*a*)
```
      READ(5, 10) I
10    FORMAT(A3)
```

(*b*)
```
      READ(5, 20) L
20    FORMAT(A7)
```

(*c*)
```
      DIMENSION K(4)
      READ(5, 30) K
30    FORMAT(4A4)
```

(*d*)
```
      DIMENSION J(3)
      READ(5, 40) J
40    FORMAT(3A5)
```

(*a*) Como el tamaño del campo es de 3, la cadena de caracteres JOH en las columnas 1-3 se asigna a la variable I. Se almacenará en I justificada a la izquierda con un blanco a la derecha, como se ve en la figura 9-1(*a*).

(*b*) Como el tamaño del campo es 7, la cadena de caracteres 'JOHN PA' en las columnas 1-7 se asigna a la variable L. Se truncará a la izquierda y solamente se almacenarán en I los cuatro caracteres más a la derecha, como se muestra en la figura 9-1(*b*).

(*c*) Como el tamaño de cada campo es 4, los datos se almacenan como se ve en la figura 9-1(*c*).

(*d*) Como el tamaño de cada campo es 5, las cadenas 'JOHN$_b$', 'PAUL$_b$' y 'JONES' se asignan a los elementos de J. Sin embargo, puesto que solamente se colocan en el almacenamiento los cuatro caracteres más a la derecha, los datos se almacenan como se muestra en la figura 9-1(*d*).

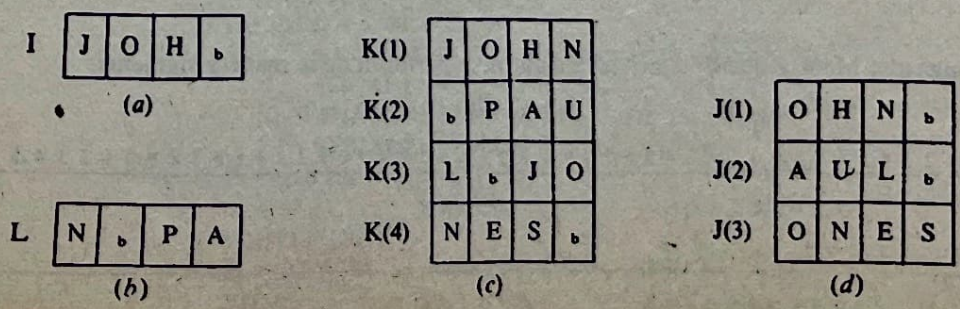


Fig. 9-1

9.3 Suponga que M=4 y que una tarjeta de datos se perfora como

```
                    1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
E R I C S O N   M O R T Y
```

(*a*) Encuentre los datos almacenados, si la instrucción de entrada es

```
      INTEGER A, B, C, D
      READ(5, 10) A, B, C, D
10    FORMAT(4A4)
```

(*b*) Encuentre la salida, si la instrucción correspondiente es

(*i*)
```
      WRITE(6, 20) A, B
20    FORMAT(1X, A8, A8)
```

(*ii*)
```
      WRITE(6, 30) C, D, B, A
30    FORMAT(1X, 2A1, 1X, 2A4)
```

(*a*) Los datos en forma de caracteres se almacenan como

A	E	R	I	C
B	S	O	N	b
C	M	O	R	T
D	Y	b	b	b

(*b*) (*i*) Excluyendo el indicativo de control de carro 1X, los tamaños de los campos para A y B son 8 y 8. Los contenidos del almacenamiento se imprimen justificados a la derecha en sus respectivos campos, como se indica enseguida:

$_{bbbb}ERIC_{bbbb}SON_{b}$

Obsérvese que $w - M = 8 - 4 = 4$ blancos preceden a ERIC y SON_b.

(*ii*) Los tamaños de los campos para C, D, B y A son respectivamente 1, 1, 4, 4. Las cadenas de caracteres en C y D se truncan a la derecha; por lo tanto, se imprime

MY_bSON_bERIC

9.4 Suponga que M=4 y que dos tarjetas de datos se perforan de la manera siguiente:

```
                    1                   2                   3                   4
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
Tarjeta 1   J E F F E R S O N           T H O M A S
Tarjeta 2   W A S H I N G T O N         G E O R G E
```

Encuentre la salida para cada uno de los segmentos de programa:

(*a*)
```
      DIMENSION NOMBRE(30)
      DO 100 I = 1, 2
            READ(5, 60) NOMBRE
   60       FORMAT(5X, 30A1)
            WRITE(6, 60) NOMBRE
  100 CONTINUE
```

(*b*)
```
      DIMENSION NOMBRE(10)
      DO 100 I = 1, 2
            READ(5, 35) NOMBRE
   35       FORMAT(5X, 10A4)
            WRITE(6, 45)
   45       FORMAT(5X, 10A3)
  100 CONTINUE
```

(*c*)
```
      DO 100 I = 1, 2
            READ(5, 25)
   25       FORMAT(5X, 30HAAAAAAAAAABBBBBBBBBBCCCCCCCCCC)
            WRITE(6, 25)
  100 CONTINUE
```

(*a*) Cada carácter se almacena en un elemento de NOMBRE y luego se imprime. Por lo tanto, la salida aparece como en la tarjeta de datos:

```
JEFFERSON      THOMAS
WASHINGTON     GEORGE
```

(*b*) Nótese que a cada elemento de NOMBRE, se asignan cuatro caracteres, pero solamente se imprimen tres caracteres, los tres primeros. Por lo tanto, la salida aparecerá como sigue:

```
JEFERSN        HOMS
WASINGON       EORE
```

(*c*) Los 30 caracteres que se imprimen son los que están en las columnas 6-35; éstos reemplazan a los caracteres que siguen a 30H en la proposición FORMAT. Por lo tanto, la salida aparecerá como en la tarjeta de datos:

```
JEFFERSON      THOMAS
WASHINGTON     GEORGE
```

9.5 Supongamos que se perfora una frase completa S en una tarjeta desde la columna 1. Cuente el número N de palabras en S.

Suponiendo que las palabras se separan por un blanco y que el final de la frase termina con un punto, N es igual al número de blancos que preceden al punto más uno. Puesto que una tarjeta contiene 80 columnas, almacenamos nuestra frase en un arreglo lineal CADENA con 80 elementos. El programa es como sigue:

```
      INTEGER CADENA(80), BLANCO, PUNTO
      DATA BLANCO/' '/, PUNTO/'.'/
      READ(5, 10) CADENA
   10 FORMAT(80A1)
      N = 0
      DO 100 K = 1, 80
C           EXAMINE SI HAY UN CARACTER BLANCO
            IF(CADENA(K).EQ.BLANCO) N = N + 1
C           EXAMINE SI ES EL FINAL DE LA FRASE
            IF(CADENA(K).EQ.PUNTO) GO TO 55
  100 CONTINUE
   55 N = N + 1
      WRITE(6, 20) N
   20 FORMAT(1X, 'EL NUMERO DE PALABRAS= ', I3)
      STOP
      END
```

9.6 Una lista de no más de 500 nombres se perfora en tarjetas a razón de un nombre por cada una. Los últimos nombres se perforan en las columnas 1-15 y los primeros en las columnas 17-32. Se adiciona una tarjeta de delimitación de datos, en la que se perforan los caracteres XXX en las columnas 1-3. Suponiendo que M=4, escriba un segmento de programa Fortran que cuente el número N de nombres y los almacene en un arreglo.

Puesto que la capacidad M=4, cada nombre requiere al menos 32/4=8 localizaciones de memoria para almacenar los 32 posibles caracteres. Por lo tanto, almacenaremos los nombres en un arreglo NAME de 500×8 con un nombre en cada fila. Además, cada vez que leamos una tarjeta, chequeamos si aparece XXX en las primeras localizaciones de almacenamiento. El programa es el siguiente:

```
      DIMENSION NAME(500, 8)
      DATA JJ/'XXX'/
      N = 0
      DO 100 K = 1, 500
            READ(5, 10) (NAME(K, L), L = 1, 8)
10          FORMAT(8A4)
            IF(NAME(K, 1).EQ.JJ) GO TO 88
            N = N + 1
100   CONTINUE
88    .....................
```

9.7 Supóngase que M=4 y que un arreglo NAME de $I \times J$ contiene un nombre por fila. Escriba un subprograma SUBROUTINE

```
ALPHA(NAME, I, J, K, L)
```

que intercambie el K-ésimo y L-ésimo nombres si no están en orden alfabético.

Ordenamos alfabéticamente los nombres como clasificaríamos sus equivalentes numéricos, excepto si los dos nombres tienen los mismos cuatro primeros caracteres, es decir,

```
NAME(K, 1) = NAME(L, 1)
```

entonces tenemos que comparar los segundos cuatro caracteres y así, sucesivamente. El programa es entonces:

```
      SUBROUTINE ALPHA(NAME, I, J, K, L)
      DIMENSION NAME(I, J)
      DO 99 N = 1, J
            IF(NAME(K, N).GT.NAME(L, N)) GO TO 50
            IF(NAME(K, N).LT.NAME(L, N)) GO TO 40
 99   CONTINUE
 40   RETURN
 50   DO 88 M = N, J
            JSAVE = NAME(K, M)
            NAME(K, M) = NAME(L, M)
            NAME(L, M) = JSAVE
 88   CONTINUE
      RETURN
      END
```

Observamos que el efecto de la proposición numerada como 50 es equivalente a

```
50   DO 88 M = 1, J
```

Dejamos al lector que encuentre la razón de esto.

9.8 Supongamos que N nombres están en la memoria en un arreglo NAME de 500 × 8, de modo que hay un nombre por cada fila. Escriba un segmento de programa que clasifique a NAME por orden alfabético.

Supongamos que $A_1, A_2, \ldots, A_N$ son los N nombres. Utilizaremos el algoritmo de clasificación por burbuja discutido en la sección 8.2, con la diferencia de que en este caso ordenaremos alfabéticamente nombres en lugar de ordenar números. Aplicando la subrutina ALPHA del problema 9.7, ordenamos alfabéticamente A_1 y A_2, luego A_2 y A_3 y así sucesivamente, hasta A_{N-1} y A_N. Después de esta "barrida" completa de los elementos, el nombre correcto estará en la última posición. Repetimos con los nombres $A_1, A_2, \ldots, A_{N-1}$. Después de barrer completamente estos nombres, el nombre correcto estará en la penúltima posición. Continuamos este procedimiento y después $N-1$ barridas, los nombres estarán en orden alfabético.

Para traducir el algoritmo a Fortran, utilizamos un ciclo DO anidado. El índice externo I variará de 1 a $N-1$ y el índice interno J desde 1 hasta $N-1$. El programa es el siguiente:

```
      NN = N - 1
      DO 99 I = 1, NN
            JJ = N - I
            DO 88 J = 1, JJ
                  CALL ALPHA(NAME, 500, 8, J, J + 1)
88          CONTINUE
99    CONTINUE
```

VARIABLES LOGICAS Y CONECTORES

9.9 Supongamos que una tarjeta de datos se perfora como sigue:

```
                    1         2         3
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
NOW IS THE TIME TO RETURN
```

Suponiendo que cualquier carácter diferente de blanco que esté en el campo, distinto de la letra T, se aceptará como.FALSO.(ver página), encuentre los valores de I, J, K, X y Y si la instrucción es

```
      LOGICAL I, J, K, X, Y
      READ(5, 10) I, J, K, X, Y
10    FORMAT(L6, L3, 3L6)
```

Los tamaños de los campos son 6, 3, 6, 6 y 6, respectivamente. El primer carácter diferente de blanco en las columnas 1-6 es N, en las columnas 7-9 es T, en las columnas 10-15 es E, en las columnas 16-21 es T y en las columnas 22-27 es T. Por lo tanto, I y K se leen como .FALSE. y J, X y Y se leen como .TRUE.

9.10 Escriba en Fortran lo siguiente:

(*a*) Si J=K y N > 15, pare.
(*b*) Si $X \geq Y^2$ ó A+B < C, vaya a la proposición número 71.
(*c*) Si A no es mayor que B, vaya a la proposición numerada como 72.
(*d*) Si es falso que A < B y C ≥ D, pare.

Insistimos en que "y", "o" y "no" se denotan en Fortran como ".AND.", ".OR." y ".NOT.", respectivamente.

(*a*) IF(J.EQ.K.AND.N.GT.15) STOP
(*b*) IF(X.GE.Y**2.OR.(A + B).LT.C) GO TO 71
(*c*) IF(.NOT.A.GT.B) GO TO 72
(*d*) IF(.NOT.(A.LT.B.AND.C.GE.D)) STOP

9.11 Supongamos que K contiene el valor 5 y A y B se han declarado como variables LOGICAS. Encuentre el valor final de A después de cada segmento de programa Fortran:

(*a*)
```
A = .FALSE.
B = 2*K.LT.15
A = A.OR.B
```

(*b*)
```
B = .TRUE.
A = 5.NE.K
A = A.AND.B
```

(*c*)
```
B = 10.LT.3*K
A = K**2.GT.4*K
IF(B) A = .NOT.A
```

(*a*) Por la línea uno, A es falso y por la línea dos, B es verdadero. Por lo tanto, de A.OR.B, el valor final de A, es verdadero.

(*b*) Por la línea uno, B es verdadero y A es falso por la línea dos. Por lo tanto, de A.AND.B, el valor final de A es falso.

(*c*) Según la línea uno, B es verdadero. Por la línea dos, A es también verdadero. Puesto que B es verdadero, se ejecuta A=.NOT.A y entonces el valor de A se cambia a falso.

9.12 Supongamos que A, B y C son variables lógicas y contienen .TRUE., .FALSE. y .TRUE., respectivamente. Encuentre la salida de cada una de las siguientes instrucciones:

(*a*)
```
      WRITE(6, 10) A, B, C
10    FORMAT(1X, 3L8)
```

(*b*)
```
      WRITE(6, 20), A, B, C
20    FORMAT(1X, L4, 3X, L4, 3X, L4)
```

En la salida, el código de formato *Lw* imprime *w* − 1 blancos seguidos de la letra T o F, según que la correspondiente variable lógica sea verdadera o falsa. Por lo tanto:

(*a*) T, F y T se imprimen en las columnas 8, 16 y 24, respectivamente.

(*b*) T, F y T se imprimen en las columnas 4, 11 y 18, respectivamente.

Problemas suplementarios

DATOS EN FORMA DE CARACTERES

9.13 Suponiendo que la capacidad de caracteres en M=4, encuentre los datos almacenados, dada la siguiente proposición:

```
DATA I/'YES'/, J/'GO NOW'/
```

9.14 Suponga que M=4 y que una tarjeta de datos se perfora como sigue:

```
                    1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
LENA  AUDREY  ERIC
```

Encuentre los datos en el almacenamiento, si la instrucción de entrada es

(*a*)
```
      READ(5, 10) J
10    FORMAT(A2)
```

(*b*)
```
      READ(5, 20) J
20    FORMAT(A7)
```

(*c*)
```
      DIMENSION K(5)
      READ(5, 30) K
30    FORMAT(5A4)
```

(*d*)
```
      DIMENSION L(4)
      READ(5, 40) L
40    FORMAT(4A5)
```

9.15 Suponga que M=4 y que LONDON está almacenado en IA e IB como sigue:

IA					IB				
	L	O	N	D		O	N	b	b

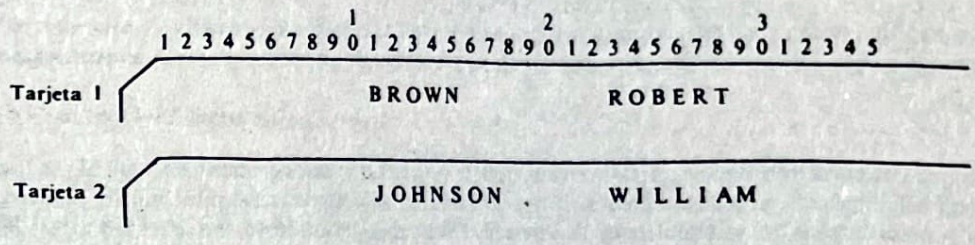
Encuentre la salida, si la proposición WRITE es

```
WRITE(6, 55) IA, IB
```

y si la proposición FORMAT acompañante es

(*a*) 55 FORMAT(1X, 2A4) (*c*) 55 FORMAT(1X, A2, A3)
(*b*) 55 FORMAT(1X, 2A7) (*d*) 55 FORMAT(1X, A3, A2)

9.16 Suponga que M=4 y que dos tarjetas de datos se perforan como sigue:



Encuentre la salida para cada uno de los siguientes segmentos de programa:

(*a*)
```
      DIMENSION NAME(6)
      DO 99 K = 1, 2
            READ(5, 15) NAME
   15       FORMAT(10X, 6A4)
            WRITE(6, 15) NAME
   99 CONTINUE
```

(*b*)
```
      DIMENSION NLAST(3), NFIRST(3)
      DO 99 K = 1, 2
            READ(5, 15) NLAST, NFIRST
   15       FORMAT(10X, 6A4)
            WRITE(6, 15) NLAST, NFIRST
   99 CONTINUE
```

(*c*)
```
      DO 99 K = 1, 2
            READ(5, 15)
   15       FORMAT(10X, 24HAAAAAAAAABBBBBBBBCCCCCCCC)
            WRITE(6, 15)
   99 CONTINUE
```

(*d*)
```
      DIMENSION NAME(6)
      DO 99 K = 1, 2
            READ(5, 15) NAME
   15       FORMAT(10X, 6A4)
            WRITE(5, 25) NAME
   25       FORMAT(10X, 6A2)
   99 CONTINUE
```

9.17 Plantee el problema 9.5, pero ahora con las palabras de la frase separadas por uno o más blancos.

9.18 Suponga que se da un paquete que tiene no más de 80 tarjetas. Cada tarjeta contiene un nombre de estudiante en las columnas 1-32 y una letra de calificación en la columna 34. Las letras de calificación válidas son A, B, C, D, F e I (incompleta). Suponiendo que se adiciona una tarjeta de delimitación de datos con los caracteres XXX perforados en las columnas 1-3, escriba un programa para

(*a*) Contar el número de estudiantes en cada clase.
(*b*) Imprimir los nombres de los estudiantes.

9.19 Suponga que un poema P se perfora en un paquete de tarjetas, de manera que cada línea del poema esté en una tarjeta y suponga también, que el paquete tiene una tarjeta de dlimitación de datos con los caracteres XXX perforados en las columnas 1-3. (Suponga que el poema P no tiene más de 20 líneas.) Escriba un programa que imprima el poema P y el número de tarjetas en el paquete, es decir, el número de líneas en P.

9.20 En el poema P del problema 9.19, escriba un programa que cuente el número de veces que las letras A, E, I, O y U aparecen en P.

9.21 En el poema P del problema 9.19, escriba un programa que cuente el número de líneas que comienzan con la palabra EL. (Suponga que cada línea del poema se inicia en la columna 1 de la tarjeta de datos.)

9.22 En el poema P del problema 9.19, escriba un programa que cuente el número de veces que aparece la palabra EL. (Suponga que la palabra EL siempre está seguida y precedida por un blanco, excepto al comienzo de una línea.)

9.23 Suponga que una corta historia SS se perfora en un paquete de tarjetas y suponga, también, que el paquete tiene una tarjeta delimitadora con los caracteres XXX perforados en las columnas 1-3. Escriba un programa que imprima la breve historia SS y el número de líneas en SS. (Suponga que la historia no tiene más de 400 líneas, excluyendo el título, que se perfora en una primera tarjeta y el nombre del autor, que se perfora en una segunda.)

9.24 En la corta historia SS del problema 9.23, escriba un programa que cuente el número de párrafos en SS. (Suponga que cada párrafo se perfora en una nueva tarjeta a partir de la columna 6; de otra manera, todas las tarjetas se perforan desde la columna 1.)

9.25 Suponga que un título con menos de 80 caracteres se perfora en una tarjeta, a partir de la columna 1. Escriba un programa que imprima el título de manera que quede centrado entre las columnas 1-80.

9.26 Suponga que una lista de N nombres (con $N \leq 500$) se almacena en un arreglo NAME de 500×8 con un nombre por cada fila. Suponga que un arreglo lineal entero PERSON(1),..., PERSON(8) contiene otro nombre. Escriba un segmento de programa para determinar si PERSON está en NAME. Si lo está, encuentre la posición L del nombre; de lo contrario, haga L=0.

9.27 Suponga que N nombres (con $N < 500$) se almacenan en un arreglo NAME de 500×8. Un nombre almacenado en el arreglo entero PERSON(1),..., PERSON(8) debe insertarse en la L-ésima fila del arreglo NAME, de manera que los nombres en las filas L, L+1,..., N se desplacen hacia abajo en una posición. Escriba un segmento de programa que lleve a cabo esta tarea.

9.28 Suponga que N nombres (con $N \leq 500$) se almacenan alfabéticamente en un arreglo NAME de 500×8. Utilice el método de búsqueda binaria para determinar si el nombre almacenado en el arreglo entero PERSON(1),..., PERSON(8) está en el arreglo NAME. Si es así, asigne .TRUE. a la variable lógica IN y halle su posición en la fila L. Si no lo es, asigne .FALSE. a la variable IN y determine la posición en la fila L en la que este nombre debe insertarse, o insértelo como en el problema 9.27.

VARIABLES LOGICAS Y CONECTORES

9.29 Suponga que las únicas variables LOGICAS en un programa son A, B, C y D. Encuentre los errores, si los hay, en cada una de las siguientes proposiciones:

(*a*) A = FALSE (*c*) C = R.GT.(A + 2.0) (*e*) F = C.AND.D
(*b*) B = Y.GT.X (*d*) D = A.OR.B (*f*) A = A.AND..TRUE.

9.30 Suponga que una tarjeta de datos se perfora como sigue:

```
          1         2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
TOM LEFT IN THE NIGHT
```

Suponiendo que el compilador considera que cualquier símbolo distinto de T es falso, encuentre la salida de cada segmento de programa:

(*a*)
```
      LOGICAL A(6)
      READ(5, 15) A
15    FORMAT(5L4)
      WRITE(6, 25) A
25    FORMAT(1X, 5L4)
```

(*b*)
```
      LOGICAL X, Y, Z
      READ(5, 15) X, Y, Z
15    FORMAT(3X, 3L4)
      WRITE(6, 25) X, Y, Z
25    FORMAT(1X, 3L4)
```

9.31 Escriba en Fortran lo siguiente:

(*a*) Si A > B y C ≤ 10, pare.
(*b*) Si J ≠ L o A es positiva, vaya a la proposición numerada como 100.
(*c*) Si 7 < X < 10, vaya a la proposición rotulada como 200.
(*d*) Si es falso que A ≤ B y X ≥ Y, pare.

9.32 Suponga que A, B y C se han declarado como variables lógicas y tienen los valores. FALSE., .FALSE. y .TRUE., respectivamente. Encuentre el valor de cada una de las expresiones lógicas.

(*a*) A.OR.C
(*b*) B.AND.C
(*c*) A.OR..NOT.C
(*d*) .NOT.A.AND.B
(*e*) .NOT.(A.AND.B)
(*f*) A.AND.B.OR..NOT.C
(*g*) A.OR.B.AND.C
(*h*) .NOT.(.NOT.B.OR.A)
(*i*) A.OR.C.AND.B.OR.C

9.33 Suponga que J contiene el valor 10 y que A y B se han declarado como variables lógicas. Encuentre el valor final de A después de que se ejecute cada uno de los segmentos de programa.

(*a*)
```
A = .TRUE.
B = J.LT.5
A = A.AND.B
```

(*b*)
```
B = .TRUE.
A = .FALSE.
IF(.NOT.A) B = .FALSE.
A = A.OR.B
```

(*c*)
```
A = J + 10.EQ.2*J
B = .TRUE.
IF(A.OR..NOT.B) J = J + 10
A = J.GT.15
```

(*d*)
```
A = .TRUE.
B = .NOT.A
A = B.OR.J**2.EQ.100
```

9.34 Suponga que J, K y L contienen los valores 10, 20 y 30, respectivamente. Encuentre el valor de cada una de las expresiones lógicas.

(*a*) 2*J.EQ.K.AND.K.LE.L
(*b*) .NOT.J.GT.K.OR.L.LT.5
(*c*) K.EQ.10.AND..NOT.J.LT.L − 15
(*d*) .NOT.(5.EQ.J − 5.AND.2*K.EQ.J + L)

9.35 Escriba una expresión lógica Fortran, para cada una de las condiciones:

(*a*) A es menor que X, Y y Z.
(*b*) A es menor que alguno de los valores X, Y o Z.
(*c*) X > 2, pero Y ≤ 3.
(*d*) A y B son positivas, o una de ellas es mayor que 100.
(*e*) 0 < X < 1 y Y ≥ Z.

Respuestas a problemas suplementarios seleccionados

9.13 YES$_b$ en I GO$_b$N en J.

9.14 (*a*) LE$_{bb}$ en J. (*b*) A$_{bb}$A en J. (*c*) LENA en K(1), $_{bb}$AU en K(2), DREY en K(3), $_{bb}$ER en K(4), e IC$_{bb}$ en K(5). (*d*) ENA$_b$ en L(1), AUDR en L(2), Y$_{bb}$E en L(3), e IC$_{bb}$ en L(4).

9.15 (*a*) LONDON$_{bb}$ (*b*) $_{bbb}$LOND$_{bbb}$ON$_{bb}$ (*c*) LOON$_b$ (*d*) LONON

9.16 (*a*), (*b*) y (*c*). La salida aparece como en las tarjetas de datos.
(*d*) BRN$_{bbb}$RORT$_{bb}$ y JOSO$_{bb}$WIIA$_{bb}$

9.29 (*a*) A=.FALSE. (*b*) No hay error. (*c*) A+2.0 no es admisible, puesto que A es una variable lógica.
(*d*) No hay error. (*e*) Inadmisible, porque F no es una variable lógica. (*f*) No hay error.

9.30 (*a*) T, F, F, T, F, T (*b*) F, T, T

9.31
```
(a) IF(A.GT.B.AND.C.LE.10.0) STOP
(b) IF(J.NE.L.OR.A.GT.0.0) GO TO 100
(c) IF(7.0.LT.X.AND.X.LT.10.0) GO TO 200
(d) IF(.NOT.(A.LE.B.AND.X.GE.Y)) STOP
```

9.32 (*a*) T (*b*) F (*c*) F (*d*) F (*e*) T (*f*) F (*g*) F (*h*) F (*i*) T

9.33 (*a*) F (*b*) F (*c*) T (*d*) T

9.34 (*a*) T (*b*) T (*c*) F (*d*) F

9.35
```
(a) A.LT.X.AND.A.LT.Y.AND.A.LT.Z
(b) A.LT.X.OR.A.LT.Y.OR.A.LT.Z
(c) X.GT.2.0.AND.Y.LE.3.0
(d) A.GT.0.0.AND.B.GT.0.0.OR.(A.GT.100.0.OR.B.GT.100.0)
(e) (0.0.LT.X.AND.X.LT.1.0).AND.(Y.GE.Z)
```

Capítulo 10

Aspectos adicionales de entrada/salida

10.1 INTRODUCCION

Hay otros tópicos de entrada/salida que no hemos discutido. En el capítulo 3, discutimos las especificaciones de campo I, F, E y X; en el capítulo 9, las especificaciones A, H y L. Este capítulo introducirá varios aspectos nuevos; además, discutiremos cómo sacar (imprimir) la gráfica de una función.

Primero, deseamos recordar al lector que el primer carácter en cualquier cadena de salida (registro) no se imprime por la impresora de línea, sino que se usa como control de carro. Este y otros materiales sobre entrada/salida se pueden encontrar en los capítulos 3 y 9.

10.2 PROPOSICION DATA

Consideremos la constante π. Para almacenar este valor, se puede usar la proposición de asignación

```
PI = 3.14159
```

Como ésta es una proposición ejecutable, la constante 3.14159 será asignada a PI cuando se ejecute la proposición. Puesto que π no se altera (constante) a través del programa, se puede fijar el valor de 3.14159 de una vez por todas. Esto se puede realizar por la proposición DATA, así:

```
DATA PI/3.14159/
```

Esto es, la palabra DATA es seguida por el nombre de la variable y luego por su valor encerrado en diagonales. Una proposición DATA no es ejecutable y, por tanto, debe colocarse antes de cualquier proposición ejecutable del programa.

También se pueden utilizar proposiciones DATA para inicializar los valores de ciertas variables; por ejemplo, podemos desear inicializar el contador KOUNT en 1 y la suma acumulada SUM en 0.0. Esto se puede hacer mediante

```
DATA KOUNT/1/
DATA SUM/0.0/
```

Se puede usar una proposición DATA para agrupar varios valores, por ejemplo,

```
DATA KOUNT/1/, SUM/0.0/
```

o también,

```
DATA KOUNT, SUM/1, 0.0/
```

La última proposición tiene la forma general:

DATA *Lista de variables/Lista de constantes/*

Aquí las variables están separadas por comas, las constantes están separadas por comas y hay una correspondencia uno a uno, entre las variables y las constantes.

Aunque la proposición DATA tiene el mismo efecto que las proposiciones de asignación, unas y otras tienen varias diferencias. La proposición DATA es una declaración y no es ejecutable, de modo que los

valores de las variables se especifican durante la compilación. Por otra parte, las proposiciones de asignación son ejecutables y, por tanto, se realizan durante la ejecución.

Nota. Hacemos énfasis en la palabra "inicializar" en nuestra discusión. Aunque podemos inicializar un contador mediante

```
DATA KOUNT/1/
```

la variable KOUNT se puede cambiar durante la ejecución. Sin embargo, puesto que la proposición DATA no es ejecutable *no se puede* llamar durante la ejecución para reinicializar la variable. En consecuencia, KOUNT tendrá el valor de 1, mientras no se le asigne otro valor durante la ejecución, pero una vez que se le asigne a KOUNT otro valor, la proposición DATA no puede ejecutarse para reinicializar a KOUNT en 1.

Se pueden también inicializar los elementos de un arreglo, utilizando la proposición DATA. En particular, siempre se puede inicializar un arreglo completo escribiendo solamente su nombre; por ejemplo,

```
DIMENSION GRADE(4)
DATA GRADE/92.0, 78.0, 43.5, 88.0/
```

almacena 92.0, 78.0, 43.5, 88.0 en GRADE(1),..., GRADE(4), respectivamente. Hacemos énfasis en que si se designa un arreglo solamente por su nombre, entonces el arreglo entero se inicializa en el orden en que es almacenado; es decir, el primer subíndice varía más rápidamente, etc. (Véase sección 6.6.) Algunos compiladores también permiten el uso de un DO implícito en una proposición DATA, para inicializar parte de un arreglo. Por ejemplo, las proposiciones

```
INTEGER A(10)
DATA A(2)/5/, (A(I), I = 4, 7)/1, 2, 3, 4/
```

almacenan el valor de 5 en A(2) y los valores 1, 2, 3, 4 en A(4), A(5), A(6) y A(7), respectivamente.

Fortran también permite un factor de repetición en las proposiciones DATA. Específicamente, la notación

```
n*
```

escrita antes de una constante, donde *n* es un entero positivo sin signo, indica que esa constante se repite *n* veces. Por ejemplo,

```
DIMENSION X(100)
DATA X/100*0.0/
```

indica que 0.0 se rèpite 100 veces, de tal forma que se asignó 0.0 a todos los 100 elementos del arreglo X.

Las proposiciones DATA también pueden utilizarse para almacenar cadenas de caracteres constantes y constantes lógicas. (Por una cadena de caracteres constantes, significamos una cadena de caracteres encerrados por apóstrofes, por ejemplo, 'PROMEDIO' y 'FIN'.)

El Fortran normal no permite que aparezcan cadenas de caracteres constantes en proposiciones de asignación. Por tanto, para almacenar caracteres sin utilizar una proposición READ, se debe usar la proposición DATA. Esto se discutió en la sección 9.2. Resumimos la discusión como sigue.

El número máximo M de caracteres que se pueden almacenar en cualquier localización de memoria (llamado la *capacidad de caracteres*) varía de máquina a máquina. Supongamos que M=4. Entonces

```
DATA NOMBRE/'PAT'/
```

almacena la cadena de caracteres PAT justificada a la izquierda en la localización NOMBRE, como

NOMBRE | P | A | T | b |

con un blanco añadido a la derecha. En el caso de que M=2, solamente los dos caracteres que están más a la izquierda se almacenarán en memoria:

NOMBRE | P | A |

Algunos compiladores, tales como WATFOR-WATFIV detectarán como error el hecho de que la longitud *l* de la cadena de caracteres exceda la capacidad M. Otros compiladores pueden almacenar el exceso de caracteres en la siguiente localización del arreglo, si se usó un nombre de arreglo para leer los caracteres. (Ver problema 10.3.)

Para almacenar valores lógicos utilizando proposiciones DATA, simplemente se escribe

```
LOGICAL A, B
DATA A, B/.TRUE., .FALSE./
```

como se discutió en la sección 9.8.

EJEMPLO 10.1

(*a*) Considerar el segmento de programa

```
DIMENSION B(100)
LOGICAL L(5)
DATA B/40*0.0, 30*1.0, 20*2.0, 10*3.0/, L/3*.TRUE., .FALSE., .TRUE./
```

Entonces B(1) a B(40) toman el valor de 0.0, B(41) a B(70) toman el valor de 1.0, y así sucesivamente. También L(1) a L(3) y L(5) toman el valor de .TRUE., pero .FALSE. se asigna a L(4)

(*b*) Las proposiciones

```
DIMENSION A(3, 2)
DATA A, B, C, D/1., 2., 3., 4., 5., 6., 3*9.9/
```

harán que

A(1, 1) = 1., A(2, 1) = 2., A(3, 1) = 3., A(1, 2) = 4.,
A(2, 2) = 5., A(3, 2) = 6. y B = 9.9, C = 9.9, D = 9.9

10.3 CAMPO T

En la misma forma que usamos el tabulado en una máquina de escribir, podemos informar al computador de la posición de la columna para leer o escribir. Esto se puede hacer utilizando el campo T. La forma general de la especificación de campo es

Tp

donde *p* es una constante entera sin signo. La *p* especifica la *posición de la columna de partida*, donde la información se va a leer o imprimir.

Supongamos, por ejemplo, que deseamos almacenar en las localizaciones I y X los valores perforados en las columnas 13-15 y en las columnas 21-28, respectivamente. Podemos realizar esto por

```
      READ(5, 80) I, X
80    FORMAT(12X, I3, 5X, F8.2)
```

Utilizando campos T, podemos utilizar la siguiente proposición FORMAT equivalente:

```
80    FORMAT(T13, I3, T21, F8.2)
```

Obsérvese que el campo T especifica la *posición de partida* para la siguiente especificación de campo.

A veces encontramos que el uso del campo T es más fácil, puesto que elimina la necesidad de contar las columnas.

Puesto que todas las especificaciones de campo se refieren a campos contiguos, la pareja READ-FORMAT

```
      READ(5, 22) A, B
22    FORMAT(T20, F8.2, 12X, F8.2)
```

implica que el valor para A se encontrará en las columnas 20-27 y el valor para B estará en las columnas 40-47.

Al utilizar campos T, los valores pueden leerse en cualquier orden. Por ejemplo,

```
      READ(5, 23) X, Y
23    FORMAT(T31, F8.2, T11, F8.2)
```

asignará a X los números de las columnas 31-38 y asignará a T los números de las columnas 11-18.

Podemos, además, leer varias veces desde la misma posición. Por ejemplo,

```
      READ(5, 24) M, N, N1, N2, N3
24    FORMAT(T21, A3, T21, I3, T21, 3I1)
```

almacenará la información de las columnas 21-23, como una cadena de caracteres (es decir, en forma alfanumérica) en la localización M, como un valor entero (en forma numérica) en la localización N y los dígitos en las localizaciones N1, N 2 y N3, respectivamente.

En salida, la especificación de campo T da la posición de partida del registro de salida(cadena) y no de la hoja de impresión. Puesto que el primer carácter en un registro de salida se usa para control de carro, la posición de partida sobre la hoja de impresión será una posición menos que el número que aparece en la especificación de campo T. Por ejemplo,

```
      WRITE(6, 28) I, J, K
28    FORMAT(15X, I7, T28, I5, T42, I9)
```

instruye al computador para que imprima I justificado a la derecha en las columnas 14-20, para que imprima J justificado a la derecha en las columnas 27-31 e imprima K justificado a la derecha en las columnas 41-49. Esto es equivalente a

```
      WRITE(6, 28) I, J, K
28    FORMAT(T15, I7, 6X, I5, 9X, I9)
```

Naturalmente, los valores se pueden imprimir en cualquier orden. Por ejemplo,

```
      WRITE(6, 39) X, Y
39    FORMAT(T31, F8.2, T11, F8.2)
```

imprimirá Y en las columnas 10-17 y X en las columnas 30-37.

10.4 CAMPO G

El formato generalizado G se puede usar para entrada/salida en datos enteros, reales, lógicos o complejos (los datos complejos se discuten en el capítulo 11.) La forma general de la especificación de campo es

Gw.d

donde *w* denota la longitud de campo y *d* es una constante entera sin signo. (Se puede también usar *rGw.d*, donde *r* denota el número de repetición.)

El significado de *Gw.d* se determina por el tipo de la variable correspondiente, como sigue:

Variable entera

Gw.d tiene el mismo significado que *Iw*, tanto en entrada como en salida.

Variable lógica

Gw.d tiene el mismo significado que *Lw*, tanto en entrada como en salida.

Variable real

Hay dos casos.

1. *Entrada.* Gw.*d* tiene el mismo significado que *Ew.d* ó *Fw.d*, según que el valor de entrada tenga o no perforada una E en su campo.
2. *Salida.* Aquí *d* denotará siempre el número de dígitos significativos. La salida estará en forma F, si el valor se puede imprimir en un campo de longitud $w-4$. En este caso, el valor será impreso exactamente con cuatro blancos a la derecha del campo (es decir, justificado a la derecha en la quinta columna desde el lado derecho del campo). La salida estará en forma E, si el valor no se puede imprimir en un campo de longitud $w-4$. En este caso, el valor se imprimirá justificado a la derecha, en su campo.

Variable compleja

Se discute en el capítulo 11.

EJEMPLO 10.2

(*a*) Supongamos que I, J, A, B se almacenan por medio de

```
      LOGICAL J
      READ(5, 30) I, J, A, B
30    FORMAT(3G10.3, G12.3)
```

donde la tarjeta de datos está perforada, como sigue:

```
          1         2         3         4
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
  1 2 3   T H E   E N D       4 . 5 6 7 8 9   - 1 2 . 3 4 5 6 E - 0 6
```

Puesto que G10.3 tiene el factor de repetición 3, las especificaciones de campo son G10.3, G10.3, G10.3, G12.3 con longitudes de campo de 10, 10, 10 y 12, respectivamente.

(*i*) El primer G10.3 corresponde a la variable entera I, por tanto se interpreta como I10. Luego el valor que se le asigna a I es 123, el entero perforado en el primer campo, en las columnas 1-10.

(*ii*) El segundo G10.3 corresponde a la variable lógica J, así que se interpreta como L10. El primer carácter en el segundo campo, las columnas 11-20, es T; por tanto, .TRUE. se almacena en J.

(*iii*) El tercer G10.3 corresponde a la variable real A. No hay una E perforada en su campo, columnas 21-30, así que tiene el mismo significado que F10.3. Luego se asigna a A el valor de 4.56789.

(*iv*) G12.3 corresponde a la variable real B. Hay una E perforada en este campo, columnas 31-42, así que llega a ser E12.3. Luego $-12.3456E-06$, o en forma equivalente -0.0000123456 se almacena en B.

(*b*) Supongamos que la instrucción de salida es

```
      DO 100 K = 1, 6
            WRITE(6, 30) A(K)
30          FORMAT(6X, G12.4)
100   CONTINUE
```

donde A es un arreglo lineal con seis elementos que contienen respectivamente,

$$77.777,\quad 0.333\times 10^{-5},\quad 2.2222\times 10^{2},\quad 666.66\times 10^{7},\quad 12.3,\quad 8.88\times 10^{-3}$$

El código G12.4 redondea cada número a cuatro cifras significativas:

77.78, 0.00000333, 222.2, 6667000000., 12.30, 0.00888

El campo de salida (columnas 6-17) tiene una longitud de 12. Aquellos números que no pueden imprimirse con cuatro espacios a la derecha, es decir, que requieren más de $w-4=8$ columnas, se imprimen en forma exponencial. Luego, A(2) y A(4) se imprimen en forma exponencial y la salida aparece como en la figura 10-1. Obsérvese que uno puede decir inmediatamente los números que están en forma exponencial.

Fig. 10-1

10.5 FACTOR DE ESCALA

Cuando se trabaja con grandes grupos de datos, algunas veces es útil incorporar un factor de escala en los campos E o F. Un factor de escala tiene la forma

sP

donde s (llamado el factor de escala) es una constante entera sin signo, o una constante entera negativa. Se escribe a la izquierda del primer campo al que se va a aplicar, por ejemplo,

3PF8.2 ó −4PE14.4

Sin embargo, el efecto del factor de escala sobre el campo F es diferente al efecto sobre el campo E.

(*a*) Factor de escala usado con el campo F

Cuando se usa con el campo F, el efecto de sP es como sigue:

Valor externo = valor interno $\times 10^{s}$

o, equivalentemente,

Valor interno = valor externo $\times 10^{-s}$

Esto es válido, tanto en entrada como en salida.

EJEMPLO 10.3

(*a*) Supóngase que X contiene el valor de 0.325 y que deseamos imprimir a X en porcentaje. Podemos escribir

```
      WRITE(6, 10) X
10    FORMAT(1X, 2PF8.2, 1X, 'PERCENT')
```

El valor interno es multiplicado por 10^2, así que las salidas serán

32.50 PERCENT

(*b*) Supóngase que el número perforado en las columnas 1-8 de una tarjeta de datos es 7.50 y que se ejecuta

```
      READ(5, 20) RATE
20    FORMAT(2PF8.2)
```

Entonces, el número asignado a RATE será 0.075.

En otras palabras, cuando se usa *sP* con el campo F en entrada, el número almacenado será el número perforado, multiplicado por un factor de 10^{-s}; pero cuando se usa para salidad, el número impreso será el número interno, multiplicado por un factor de 10^{s}.

(*b*) Factor de escala usado con el campo E

Cuando se usan los factores de escala con el campo E en entrada, éstos no se tienen en cuenta. En salida, la magnitud del número impreso no cambia, pero la mantisa es multiplicada por un factor de 10^{s} y el exponente es reducido por *s*. (Recordamos que la mantisa es un número entre 0.1 y 1.0 ó -0.1 y -1.0.)

Por ejemplo, supongamos que el valor interno de X es 0.0004321. Si imprimimos a X utilizando la especificación de campo E12.4; entonces, X aparecerá en la forma

```
0.4321E - 03
```

Por otra parte, si imprimimos X utilizando la especificación de campo 1PE12.4, entonces X aparecerá en la forma

```
4.321E - 04
```

(Ahora el valor de X aparece en *notación científica*; es decir, como un número entre 1.0 y 10.0 ó -1.0 y -10.0 seguido de una potencia de 10.)

Advertencia. Una vez que un factor de escala se usa en un campo E a F, automáticamente se *aplica a todos los campos sucesivos E ó F* en el programa hasta que se encuentre otro factor de escala. Por tanto, se debe usar el factor de escala

```
0P      (ceroP)
```

cuando se desea eliminar el efecto de un factor de escala precedente. Por ejemplo, supongamos que los valores internos de A, B, C y D son 0.0111, 0.0222, 0.0333 y 0.0444, respectivamente, y que las instrucciones de salida son como sigue:

```
      WRITE(6, 10) A, B
10    FORMAT(1X, 1PE12.4, 5X, E12.4)
      WRITE(6, 20) C, D
20    FORMAT(1X, E12.4, 5X, 0PE12.4)
```

Entonces, el factor de escala 1P se aplica para imprimir A, B y C, pero no para D o cualquier valor real en cualquier proposición WRITE sucesiva. Luego, la salida aparecerá en la forma

```
1.110E-02     2.220E-02
3.330E-02     0.4440E-01
```

10.6 REGLA DEL PARENTESIS IZQUIERDO

Recordemos que si la lista de variables en una proposición READ o WRITE contiene más elementos que las especificaciones de campo que hay en la proposición FORMAT correspondiente, entonces la proposición FORMAT se repite una y otra vez (leyendo una nueva tarjeta de datos o imprimiendo una nueva línea cada vez), hasta que todas las variables sean acomodadas. Por ejemplo, supongamos que se ejecuta la siguiente:

```
      READ(5, 10) A, B, C, D, E, F, G, H
10    FORMAT(F6.2, F8.3, F5.1)
```

Entonces la proposición FORMAT se usa para leer valores para A, B y C (de la primera tarjeta de datos); luego, se repite la proposición FORMAT para D, E y F (de la segunda tarjeta de datos); luego, se repite para G y H (de la tercera tarjeta de datos).

Algunas veces podemos desear que se repita solamente una parte de una proposición FORMAT. La regla es que la proposición FORMAT se repite empezando en el paréntesis izquierdo que esté más a la

derecha y continuando hasta el paréntesis final de la proposición FORMAT. Por ejemplo, supongamos que la proposición FORMAT anterior se modifica, como sigue:

```
      READ(5, 20) A, B, C, D, E, F, G, H
20    FORMAT(F6.2, (F8.3, F5.1))
```

Después que la proposición FORMAT se use para A, B y C, se repite la proposición, pero solamente empezando en el paréntesis izquierdo que está más a la derecha, es decir, empezando con la especificación de campo F8.3. Por tanto, D y E se leen de la segunda tarjeta de datos; F y G de la tercera tarjeta y H de la cuarta tarjeta de datos, utilizando cada vez las especificaciones F8.3 y F5.1. La regla anterior también se aplica a un paréntesis que esté precedido por un factor de repetición.

Consideremos por ejemplo,

```
      READ(5, 30) A, B, C, D, E, F, G, H
30    FORMAT(F6.2, 2(F8.3, 3X), F5.1)
```

La proposición FORMAT es equivalente a la proposición

```
30    FORMAT(F6.2, (F8.3, 3X, F8.3, 3X), F5.1)
```

Después que la proposición FORMAT se usa para A, B, C y D, la proposición FORMAT se repite desde el paréntesis izquierdo que está más a la derecha, es decir, desde F8.3 y continúa hasta el paréntesis final de la proposición (no hasta el final del paréntesis interno). Luego, E, F y G toman valores de la segunda tarjeta de datos, utilizando las especificaciones F8.3, 3X, F8.3, 3X y F5.1; H adopta un valor de la tercera tarjeta de datos, utilizando la especificación F8.3.

La regla anterior también se aplica a proposiciones FORMAT que acompañan a proposiciones WRITE.

10.7 FORMATO A TIEMPO DE EJECUCION

De acuerdo con hechos discutidos posteriormente, una vez que un programa está escrito, todas las proposiciones FORMAT están especificadas y, por tanto, fijas. Sería muy conveniente que se pudieran modificar las proposiciones FORMAT durante la ejecución para fijar el dato; por ejemplo, se pueden agrandar las longitudes de campo si los valores son mayores de lo previsto. Esta sección discute esta capacidad, utilizando las proposiciones FORMAT llamadas de *tiempo de ejecución* (*especificado por el usuario*).

Como lo sugiere el nombre, una proposición FORMAT que está dada (especificada) durante la ejecución es llamada una proposición FORMAT a tiempo de ejecución. La idea general en el fondo de este formato es almacenar (entrada) los formatos de las especificaciones de campo, como una cadena de caracteres en un arreglo. La proposición READ/WRITE llama al arreglo cuando las especificaciones de campo del arreglo vayan a ser usadas para entrada/salida.

Consideremos el par de proposiciones

```
      READ(5, 10) A, B, I, J
10    FORMAT(2(F10.2, 2X), I5, 2X, I8)
```

Suponiendo que la capacidad de caracteres es M=4, mostraremos cómo se puede obtener el mismo resultado utilizando una proposición FORMAT a tiempo de ejecución. Primero, perforamos en una tarjeta de datos

```
(2(F10.2, 2X), I5, 2X, I8)
```

y luego, leemos la cadena de caracteres (utilizando campo A) en un arreglo. Es decir,

```
      INTEGER FORM(6)
            :
      READ(5, 20) FORM
20    FORMAT(6A4)
```

Los caracteres almacenados en el arreglo FORM están como sigue:

FORM(1)	(	2	(	F
FORM(2)	1	0	.	2
FORM(3)	,	2	X	)
FORM(4)	,	I	5	,
FORM(5)	2	X	,	I
FORM(6)	8	)		

Para usar este formato a tiempo de ejecución, escribimos

```
READ(5, FORM) A, B, I, J
```

Obsérvese que el nombre del arreglo FORM (donde se han almacenado las especificaciones de campo) está escrito en lugar de el rótulo de una proposición FORMAT.

El arreglo utilizado para formato de especificaciones puede tener cualquier número de elemento; tantos como los que se necesiten para contener las especificaciones anticipadas. Para mayor flexibilidad, frecuentemente se usa

```
      INTEGER FORM(80)
      READ(5, 40) FORM
40    FORMAT(80A1)
```

Esto acomoda cualquier especificación perforada en una tarjeta de datos y cualquier capacidad M.

Resumamos los puntos más importantes por recordar.

1. El formato especificado por el usuario debe estar almacenado en un arreglo, aunque pueda ser almacenado en una sola localización de memoria. Por ejemplo, aunque

   ```
   (1X, I8)
   ```

   puede almacenarse en una localización INFOR donde la capacidad de caracteres sea M=9, se debe aún declarar a INFOR como un arreglo.

2. Cuando se prepara la tarjeta de datos para el formato codificado, la palabra FORMAT no es perforada, pero las especificaciones de campo son perforadas dentro de paréntesis.

3. Cuando se usa el FORMAT especificado por el usuario, el nombre del arreglo que contiene las especificaciones de campo debe aparecer en la proposición READ/WRITE. Por ejemplo,

   ```
   WRITE(6, INFOR) A, B, C, J, K
   ```

   le dice al computador que las especificaciones del formato están contenidas en el arreglo INFOR.

En lugar de leer el formato de las especificaciones en un arreglo como anteriormente se podría también inicializar una proposición FORMAT a tiempo de ejecución utilizando una proposición DATA:

```
INTEGER FORM(3)
DATA FORM/'(F8.', '3,I8', ')'/
```

Si el compilador permite que cadenas de caracteres constantes aparezcan en proposiciones de asignación, entonces se podría usar

```
INTEGER FORM(3)
FORM(1) = '(F8.'
FORM(2) = '3,I8'
FORM(3) = ')'
```

(Naturalmente, podría usarse también el campo Hollerith.)

Una vez que la especificación de formato esté almacenada en un arreglo, como antes, se pueden también modificar las especificaciones del formato, dependiendo de los datos generales. Ilustremos esta técnica con ejemplos.

1. Supongamos que el formato es pre-especificado mediante

```
DIMENSION INFOR(3), X(100)
DATA INFOR/'(I3,', '2X,F', '6.2)'/, IBIG/'8.2)'/, ISMALL/'6.2)'/
```

Esto es, (I3, 2X, F6.2) está almacenado en INFOR. El segmento de programa

```
      DO 100 K = 1, 100
            IF(ABS(X(K)).GT.100.0) INFOR(3) = IBIG
100   CONTINUE
      WRITE(6, INFOR) (N, X(N), N = 1, 100)
```

chequeará primero, si la magnitud de cualquier elemento en el arreglo X no excede de 100. Si esto es así, la especificación de formato F6.2 se cambia a F8.2 para permitir más espacio para acomodar el dato. Alternativamente,

```
      DO 200 K = 1, 100
            IF(ABS(X(K)).GT.100.0) INFOR(3) = IBIG
            WRITE(6, INFOR) K, X(K)
            INFOR(3) = ISMALL
200   CONTINUE
```

cambia F6.2 a F8.2 solamente para aquellos elementos individuales del arreglo que tienen magnitud mayor que 100. Para todos los otros elementos, se retiene F6.2.

2. Supongamos que se prefiere el campo F para imprimir el arreglo A y que se usa la especificación F12.4. Sin embargo, cuando la magnitud de A(K) es muy grande para caber en 12 columnas, la impresión en forma E también es aceptable. El segmento de programa siguiente hace exactamente eso.

```
      INTEGER XFORM, EFORM, FFORM
      DIMENSION A(100), XFORM(4)
      DATA XFORM/'(1X,', 'I3,2', 'X,F1', '2.4)'/, EFORM/'X,E1'/, FFORM/'X,F1'/
                  ⋮
      DO 300 K = 1, 100
            IF(ABS(A(K)).GT.1.0E6) XFORM(3) = EFORM
            WRITE(6, XFORM) K, A(K)
            XFORM(3) = FFORM
300   CONTINUE
```

Nótese que la especificación de formato original es (1X, I3, 2X, F12.4), pero se cambia la F por E cuando A(K) es muy grande; es decir, su magnitud es mayor que 10^6.

10.8 GRAFICACION

Además de utilizar el computador para calcular listas de valores, se puede también usar para dibujar los valores en forma gráfica. La gráfica consiste de líneas de puntos "....." para los ejes y una cruz "X" para cada punto calculado (x, y) de la gráfica. Dibujando un número suficiente de puntos en la gráfica, obtenemos una imagen del gráfico. Esta técnica se ilustra abajo.

Supongamos, por ejemplo, que deseamos dibujar la función

$$y = 2x^3 - x^2 - 22x + 21$$

para $-4 \leq x \leq 4$. Debemos escoger primero los valores de x entre -4 y 4 para calcular y, digamos

$$x = -4, -3.9, -3.8, \ldots, 3.9, 4$$

Obsérvese que hay 81 valores de x y que entre ellos hay un incremento de 0.1. Requerimos también el rango de y para esos valores. Supongamos que están dados de tal forma que para $-4 \leq x \leq 4$, tenemos $-50 \leq y \leq 50$. Hacemos que cada posición a lo largo del eje y denote valores incrementales de y. Si escogemos el incremento 1.0, entonces las posiciones a lo largo del eje y representan los valores

$$y = -50, -49, -48, \ldots, 49, 50$$

Obsérvese que hay 101 de tales valores de y.

De aquí en adelante hay dos caminos para proseguir:

Método 1

Escogemos el eje horizontal (a través de la página de impresión) como el eje x, el eje vertical (hacia abajo de la página de impresión) como el eje y. En este caso, se deben calcular los valores (x, y) y almacenar el gráfico completo en una matriz G antes de que la gráfica sea dibujada. La razón para hacer esto es que la impresora imprime una línea a la vez, y es posible que dos valores diferentes de x tengan el mismo valor (horizontal) de y. (No es posible mover el carro de impresión hacia arriba y hacia abajo.) Por tanto, G tendría 101 filas correspondientes a las 101 posiciones de y, y 81 columnas correspondientes a los 81 valores de x. Esto es, G será una matriz de 101×81. La desventaja más severa de este método es que la gráfica requiere $101 \times 81 = 8181$ celdas de memoria. Por esta razón, generalmente se dibuja la gráfica, utilizando el método 2.

Método 2

Escogemos el eje horizontal como el eje y, y el eje vertical como el eje x. (El lector puede voltear la página para mirar normalmente la gráfica.) Puesto que cada valor de x da un valor único de y, podemos calcular y dibujar nuestro gráfico imprimiendo una línea a la vez. En consecuencia, se requerirá solamente un arreglo lineal de 101 celdas de memoria.

Específicamente, tenemos los siguientes pasos:

1. Hacer x igual a su valor mínimo.
2. Encontrar el valor correspondiente de y.
3. Escalar el valor de y a un entero J entre 1 y 101.
4. Imprimir una línea con una cruz (X) en la columna J de la gráfica.
5. Incrementar el valor de x y repetir los pasos 1 a 4 en tanto que x no excede su valor máximo.

Los pasos anteriores son directos. En el paso 4, también deseamos imprimir un punto (.) para la columna que representa el eje x (cuando $y=0$) y deseamos imprimir también una línea sucesiva de puntos (....) cuando la línea representa el eje y (cuando $x=0$). Esta tarea es detallada pero no difícil.

La función de escala en el paso 3 se obtiene como sigue. Está dado que y está en el rango de -50 a 50. Sumando 51 a y da un número real positivo entre 1 y 101. Puesto que deseamos redondear a y a un entero entre 1 y 101, en lugar de truncarlo, debemos también añadir 0.5 a y. Esto conduce a la función de escalización

$$\text{JSCALE(Y)} = \text{INT(Y + 51.0 + 0.5)}$$

Obsérvese que JSCALE(0)=51, es decir, la columna 51 de la gráfica es el eje x.

El programa que grafica nuestra función es como sigue:

```
      INTEGER BLANK, DOT, CROSS, LINE(101), LINEY(101)
      DATA BLANK, DOT, CROSS, LINE, LINEY/' ', '.', 'X', 101*' ', 101*'.'/
C
C     EMPIECE NUEVA PAGINA Y SALTE LINEAS
C
      WRITE(6, 10)
   10 FORMAT('1', 10X,'GRAFICA DE UNA FUNCION'//3X, 'X', 6X, 'Y'//)
C
C     EMPIECE DANDO VALORES A  X
C
      X = -4.0
   50 Y = ((2.0*X - 1.0)*X - 22.0)*X + 21.0
      J  = INT(Y + 51.5)
C           PRUEBE SI ES EL EJE X
      IF(ABS(X).LT.0.001) GO TO 100
      LINE(51) = DOT
      LINE(J) = CROSS
      WRITE(6, 20) X, Y, LINE
   20 FORMAT(1X, 2(F6.2, 1X), 101A1)
      LINE(J) = BLANK
      GO TO 200
  100 LINEY(J) = CROSS
      WRITE(6, 20) X, Y, LINEY
  200 X = X + 0.1
      IF(X.LE.4.0) GO TO 50
      STOP
      END
```

Nótese que usamos una proposición DATA para almacenar los caracteres ' ' (blanco), '.' (punto), y 'X' (cruz) en BLANK, DOT y CROSS, respectivamente. Utilizamos LINE para imprimir cada línea del gráfico, excepto cuando la línea representa el eje *y*, en cuyo caso usamos LINEY. Por tanto, LINE se inicializa con blancos, pero LINEY con puntos. (Podríamos haber usado un arreglo, pero entonces el arreglo tendría que llenarse con blancos o puntos después de cada incremento de X.) La línea

```
LINE(J) = BLANK
```

es trascendental, ya que borra la cruz en la LINE(J) después de que se imprime. Nótese que podríamos haber usado un ciclo DO para los diferentes valores de X. La salida del programa aparece en la figura 10.2. Observe que los valores de X e Y aparecen sobre el lado izquierdo de la gráfica.

Nota. La función de escala JSCALE(Y), discutida anteriormente y en el problema 10.16, requiere que se conozca con anterioridad el rango de valores de *y*. En el caso que no se tenga esta información, se puede proceder como sigue: (1) Calcular todos los valores de *y* y almacenarlos en un arreglo Y (antes de graficar nada). (2) Encontrar los valores máximo y mínimo del arreglo Y. (3) Usar esos valores para obtener la función de escala JSCALE como en el problema 10.16. (4) Dibujar la gráfica, usando la función JSCALE y el arreglo Y. (Este procedimiento también se describe en los problemas 10.36 y 10.37.)

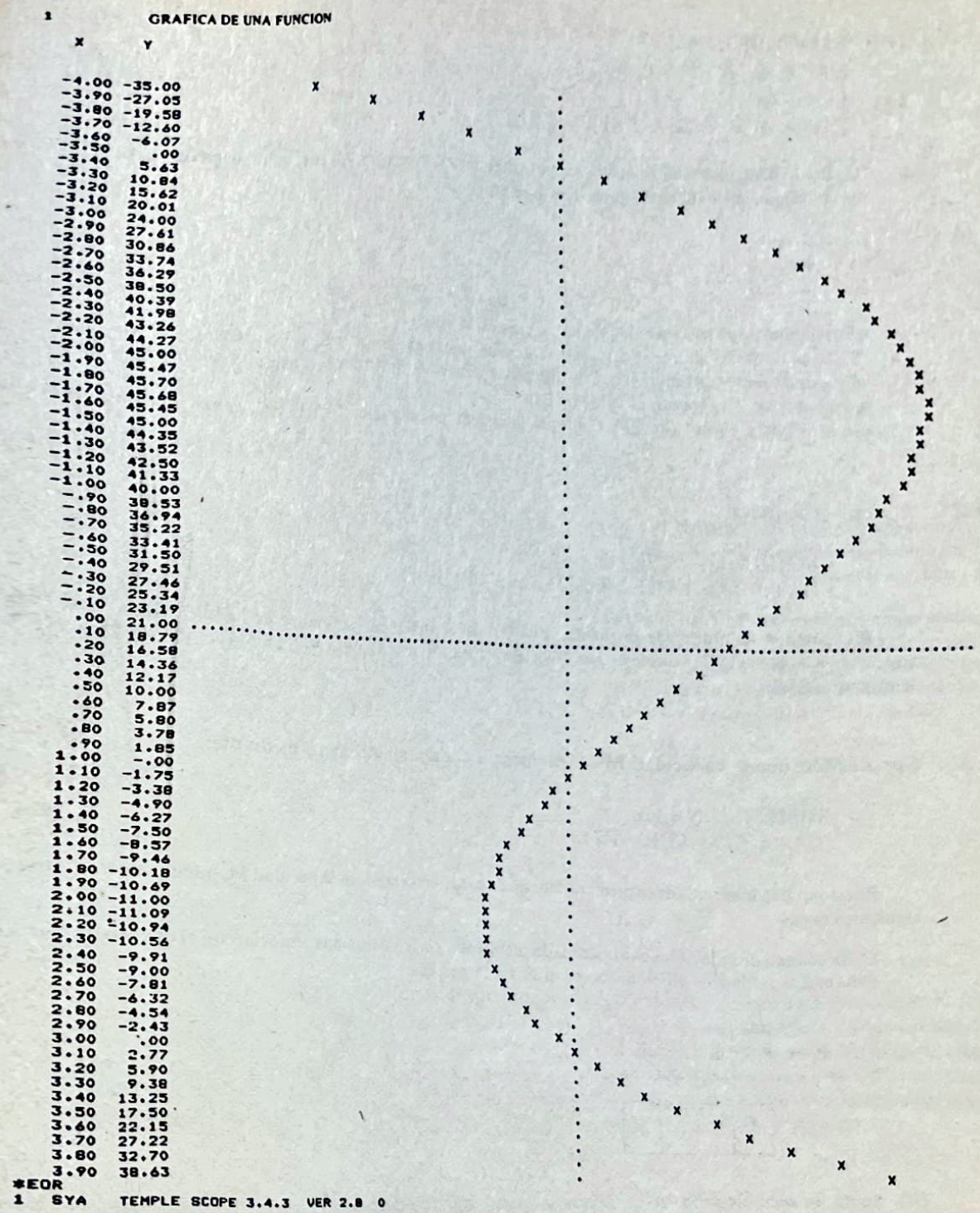


Fig. 10-2

Problemas resueltos

PROPOSICION DATA

10.1 **Encontrar la entrada:**

(*a*) DATA A, B, C /2.4, '2.4', 1.5/
(*b*) DATA A, B /2*2.4/, C/1.5/

(*c*) DIMENSION A(10)
DATA A, B, C/6*1.0, 6*2.0/

(*d*) LOGICAL C
DATA A, B, C /2.4, 'TRUE', .TRUE./

(*a*) A, B y C tienen los valores que estan entre diagonales. Por tanto, 2.4 es el valor asignado a A, 1.5 a C, y a B se le asigna una cadena de caracteres como sigue:

B | 2 | . | 4 | b |

(Suponemos que la capacidad de caracteres es M=4.)

(*b*) El 2* es un factor de repetición; por tanto, 2.4 es asignado a A y B, y 1.5 a C.

(*c*) El 6* es un factor de repetición. Los primeros 6 elementos de A toman el valor de 1.0; B y C, y los últimos 4 elementos de A, adoptan el valor de 2.0.

(*d*) Se le asigna a A un valor 2.4, B adopta la cadena de caracteres TRUE y C, el valor lógico .TRUE..

10.2 Encontrar la salida:

```
LOGICAL A(3, 2)
DATA A/2*.TRUE., 3*.FALSE., .TRUE./
```

El arreglo A se ordena de tal forma, que el primer subíndice (subíndice de la fila) varía más rápido. Entonces, a A(1, 1) y A(2, 1), se le asigna .TRUE. y .FALSE. se le asigna a A(3, 1), A(1, 2) y A(2, 2); a A(3, 2) se le asigna .TRUE..

10.3 Suponiendo que la capacidad M=4, encuentre el dato almacenado mediante:

```
DIMENSION L(2)
DATA L/'SLEEP', 'IN'/
```

Puesto que la longitud / de la primera cadena de caracteres excede la capacidad M, puede suceder una de las siguientes cosas:

(*i*) Cada cadena de caracteres es almacenada justificada a la izquierda en memoria, con el exceso de caracteres eliminados o blancos adicionales, añadidos a la derecha:

L(1) | S | L | E | E |

L(2) | I | N | b | b |

(*ii*) Se da un mensaje de error.

(*iii*) Puesto que el arreglo llamado L es utilizado en la proposición DATA, SLEEP se almacena justificado a la izquierda en L(1) y cualquier exceso de caracteres es movido a la localización L(2), como sigue:

L(1) | S | L | E | E |

L(2) | P | I | N | b |

(Insistimos en el hecho de que todos los compiladores producen el mismo resultado si ninguna longitud de la cadena de caracteres excede la capacidad M.)

CAMPO T

10.4 Supongamos que se perfora una tarjeta de datos como sigue:

```
          1         2         3
1234567890123456789012345678901234567
1111 2222 THE END     3333
```

Encontrar la salida, si se ejecuta lo siguiente:

```
(a)       READ(5, 10) I, J
     10   FORMAT(T6, I4, T3, I6)
(b)       LOGICAL J
          READ(5, 20) I, J, X, Y
     20   FORMAT(T11, A4, T11, L4, T21, F5.2, T3, F6.3)
```

(*a*) Se le asigna a I el entero que está en las columnas 6-9, esto es, en el campo de longitud 4, empezando en la columna 6. Por tanto, I adopta el valor 2222.; a J se le asigna el entero de las columnas 3-8, esto es, el que está en campo de longitud 6, empezando en la columna 3. Puesto que los blancos se interpretan como ceros en los campos numéricos, T adopta un valor de 110222.

(*b*) A I se le asigna la cadena de caracteres 'THE$_b$', los caracteres en el campo de longitud 4 empezando en la columna 11. A J se le asigna .TRUE., puesto que J es una variable lógica y T es el primer carácter diferente de blanco en el campo de longitud 4, empezando en la columna 11. (Obsérvese que I y J usan la tarjeta de datos perforada en las columnas 11-14.) Puesto que T21 y T3 instruyen a la lectora de tarjetas para ir a las columnas 21 y 3 respectivamente, a X se le asigna 333.30 y a Y se le asigna el valor 110.222.

10.5 Encontrar la salida, si se ejecuta el siguiente programa:

```
     I = 111
     J = 222
     K = 333
     WRITE(5, 30) I, J, K
30   FORMAT(T15, I3, T6, I6, T20, I8)
```

En salida *Tp* se refiere a la posición *p* en la cadena de salida (registro); por tanto, *Tp* se refiere a la columna $p-1$ en la página de impresión, puesto que no se imprime el primer carácter del registro. En consecuencia, I se imprime justificado a la derecha en el campo de longitud 3, empezando en la columna 14 de la página de impresión, es decir, en las columnas 14-16; J se imprime justificado a la derecha en el campo de longitud 6, empezando en la columna 5, es decir, en las columnas 5-10; y K se imprime justificado a la derecha en el campo de longitud 8, empezando en la columna 19; es decir, en las columnas 19-26. Ver figura 10-3.

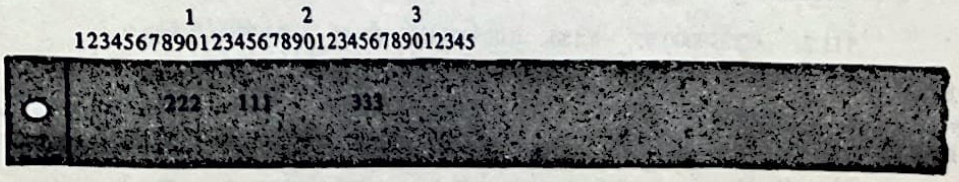
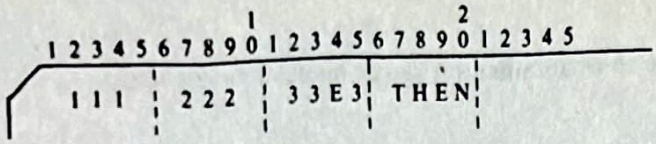


Fig. 10-3

CAMPO G

10.6 Supongamos que una tarjeta de datos se perfora como sigue:



Encontrar la salida si se ejecuta lo siguiente:

```
      LOGICAL Z
      READ(5, 10) J, A, B, Z
10    FORMAT(4G5.1)
```

Hay 4 entradas G5.1, una para cada variable. Por tanto, hay 4 campos, cada uno de longitud 5.

(*a*) J es una variable entera, luego G5.1 se interpreta como I5. Entonces, J adopta el valor 1110, el entero que aparece en las columnas 1-5. (El blanco que aparece en la columna 5 se interpreta como 0.)

(*b*) A es una variable real y no hay una E perforada en las columnas 6-10. Luego G5.1 se interpreta como F5.1. No hay punto decimal en el campo, luego se presume un decimal a la derecha del campo, por tanto a A se le asigna el valor de 222.0.

(*c*) B es una variable real y hay una E perforada en las columnas 11-15. Luego G5.1 se interpreta como E5.1. No hay punto decimal en el campo, luego se presume un decimal colocado a la izquierda de E. En consecuencia, 3.3E3 ó, equivalentemente, 3300, se asigna a B.

(*d*) Z es una variable lógica, así que G5.1 se interpreta como L5. El primer carácter diferente de blanco en las columnas 16-20 es T; por tanto, .TRUE. se asigna a Z.

10.7 Supongamos que J contiene el valor 2345 y K es una variable lógica que contiene el valor .FALSE. Encontrar la salida, si

```
      WRITE(6, 20) J, K
20    FORMAT(1X, 2G10.3)
```

Excluyendo la instrucción de control de carro, 1X, las longitudes de los campos son de 10 para J y K. Puesto que J es una variable entera, G10.3 llega a ser I10 y así, 2345 se imprime justificado a la derecha en las columnas 1-10. Como K es una variable lógica, G10.3 llega a ser L10 y así, F se imprime en la columna 20.

10.8 Supongamos un arreglo lineal A con 7 elementos que contienen, respectivamente,

111.18, −22227000.0, −6.15, 0.0000033333, 44.4, −0.023, 888.88

Encontrar la siguiente salida:

```
      DO 100 K = 1, 7
            WRITE(6, 40) A(K)
40          FORMAT(1X, G12.4)
100   CONTINUE
```

G12.4 redondea cada número a 4 cifras significativas como sigue:

111.2, −22230000.0, −6.150, 0.000003333, 44.40, −0.02300, 888.9

Aquellos números que no puedan imprimirse en campos de longitud $w-4=12-4=8$ (es decir, los que son relativamente grandes o pequeños en valor absoluto) se imprimen en forma exponencial. Los otros números se imprimen en forma F justificados a la derecha pero finalizando con 4 blancos como sigue:

```
 111.2
-0.2223E 08
 -6.150
 0.3333E-05
 44.40
-0.02300
 888.9
```

FACTOR DE ESCALA

10.9 Encontrar el valor interno que dé el siguiente código de formato y el número externo dado como entrada:

	Código de formato	Número externo dado como entrada
(*a*)	3PF6.1	123.45
(*b*)	−4PF6.1	123.45
(*c*)	4PE6.1	123.45
(*d*)	−3PE8.1	123.45E1
(*e*)	1PF6.2	12345.
(*f*)	−1PE9.2	12345E − 2

El factor de escala *sP* tiene el siguiente efecto sobre los números externos dados como entrada, escritos sin exponente:

$$\text{Número interno} = \text{número externo} \times 10^{-s}$$

El factor de escala no afecta los números escritos con un exponente como en (*d*) y (*f*). Por tanto,

(*a*) 0.12345 (*b*) 1234500. (*c*) 0.012345 (*d*) 123.45E1 ó 1234.5 (*e*) 1234.5
(*f*) Se supone un punto decimal entre 3 y 4 luego nos resulta 123.45E − 2 ó 1.2345. El factor de escala no tiene efecto sobre el número.

10.10 Supongamos que 12345.6 es el número interno. Encontrar la salida, si la especificación de campo es

(*a*) 2PF12.1 (*c*) −1PF12.2 (*e*) −1PE15.6
(*b*) −3PF12.2 (*d*) 2PE12.4 (*f*) 1PE15.3

Recordemos primero, que *Fw.d* redondea el número a *d* cifras decimales, mientras que *Ew.d* redondea el número a *d* dígitos significativos.

Para el campo F, el factor de escala *sP* multiplica el número interno por 10^s. Por consiguiente, tenemos

(*a*) 1234560.0 (*b*) 12.35 (*c*) 1234.56

Con el campo E, el factor de escala no cambia el valor del número, como lo hizo con el campo F, sino que solamente cambia la forma exponencial. Por tanto, el punto decimal se mueve multiplicando la mantisa por 10^s, y entonces se suma $-s$ al exponente. Luego

(*d*) 12.35E + 03 (*e*) 0.0123456E + 06 (*f*) 1.23E + 04

10.11 Supongamos que A contiene 111.888. Encontrar la salida, si la instrucción es

```
      WRITE(6, 10) A, A, A, A, A
10    FORMAT(1X, F10.2, E12.4, 1PF10.2, E12.4, F10.2)
```

El factor de escala *sP* se aplica a cada especificación de campo sucesiva, hasta que se encuentre otro factor de escala. Por tanto, la salida es

111.89 0.1119E + 03 1118.88 1.119E + 02 1118.88

donde los números se imprimen justificados a la derecha en campos de longitud 10, 12, 10, 12, 10, respectivamente.

10.12 Supongamos que en el problema 10.11 solamente deseamos que el factor de escala sea aplicado a la tercera A y no a la cuarta ni a la quinta. ¿Cómo se podría escribir la proposición FORMAT?

Se escribe 0P cuando se desea que el factor de escala no se aplique más. Luego, la proposición FORMAT se escribirá como sigue:

```
10  FORMAT(1X, F10.2, E12.4, 1PF10.2, 0PE12.4, F10.2)
```

FORMATO A TIEMPO DE EJECUCION

10.13 Suponiendo que la capacidad de caracteres es M=4, describa la entrada

```
INTEGER FORM(6)
DATA FORM/'(1X,', 'I2,', '5X,', 'F6.', '2)', ' ')
```

El arreglo FORM es almacenado como sigue:

FORM(1)	(	1	X	,
FORM(2)	I	2	,	
FORM(3)	5	X	,	
FORM(4)	F	6	.	
FORM(5)	2	)		
FORM(6)				

Observe que FORM(6) tiene solamente blancos.

10.14 Supongamos que J y R son arreglos lineales con 4 elementos y que tienen valores

11, 2222, 77, 666 y 3333.33, 44.44, 5.555, 88888.888

respectivamente. Encontrar la salida, si se ejecuta lo siguiente, utilizando el arreglo FORM en el problema 10.13.

```
      DATA K1/'I4,'/, K2/'F9.'/
      DO 100 I = 1, 4
            IF(J(I).GE.100) FORM(2) = K1
            IF(R(I).GE.1.0E3) FORM(4) = K2
            WRITE(6, FORM) J(I), R(I)
100   CONTINUE
```

Si J(I) ≥ 100, entonces su longitud de campo está extendido de 2 a 4, y si R(I) ≥ 1000, entonces su longitud de campo se extiende de 6 a 9. Por tanto, la salida aparecerá como en la figura 10-4. En todos los casos, el valor se redondea a 2 cifras decimales.

```
123456789012345678901234 5
11        3333.33
2222       44.44
77          5.56
666        88888.89
```

Fig. 10-4

GRAFICAS

10.15 Supongamos que deseamos usar un ciclo DO para obtener los valores de x de un entero J a un entero K con incrementos de 1/N, donde N es un entero positivo. Encontrar (*a*) el número de valores de x; (*b*) el valor de x en términos del índice INDEX del ciclo DO; (*c*) el valor de INDEX, si los hay, que da el eje y. Pruebe el resultado para el caso de J=−2, K=8 y N=4, esto es, donde x toma los valores desde −2 a −8 con incrementos de 1/4=0.25.

(*a*) Hay K − J intervalos unitarios entre J y K y N valores de x, que corresponden a cada unidad. También hay un valor de partida de x. Luego, hay un total de

$$N*(K - J) + 1$$

valores de x. En el ejemplo, x toma los valores de

$$-2, -1.75, -1.50, \ldots, 0, \ldots, 7.75, 8$$

Por tanto, hay 4(8+2)+1=41 valores de x.

(*b*) La relación entre el INDEX y X es como sigue:

x:	J	J + 1/N	J + 2/N	J + 3/N	...	K
INDEX:	1	2	3	4	...	N*(K − J) + 1

Observe que

$$x = J + (INDEX - 1)/N$$

En el ejemplo

$$x = -2 + (INDEX - 1)/4$$

(*c*) El eje y ocurre cuando x=0. Pero x puede ser igual a cero solamente si $J \leq 0 \leq K$. En este caso, hay N*(−J) valores de x que preceden a x=0. Luego, el eje y ocurre cuando INDEX=N*(−J)+1. En el ejemplo, el eje y ocurre cuando INDEX=4(2)+1=9.

10.16 Supongamos que están dados todos los valores de y que se encuentran entre los enteros L y M (donde $L < M$). Encontrar la función de escala que relaciona cada valor de y a un entero J entre 1 y 101. (Suponemos que el gráfico tendrá 101 columnas.) Pruebe la función de escala para el caso en que L=−10 y M=30, esto es, que los valores de y están entre −10 y 30.

Sumando −L a y, da valores entre 0 y M − L. Multiplicando por 100/(M − L) nos da valores entre 0 y 100. Añadiendo 1 da valores entre 1 y 101. Puesto que deseamos redondear a y a un valor entero J entre 1 y 101, en lugar de truncarlo, debemos sumar también 0.5. Luego, la función de escala es como sigue:

```
JSCALE(Y) = INT((Y + FLOAT(-L))*(100.0/FLOAT(M - L)) + 1.5)
```

En el ejemplo,

```
JSCALE(Y) = INT((Y + 10.0)*(100.0/40.0) + 1.5)
```

10.17 Escribir un programa que grafique la parábola

$$y = 3x^2 - x - 8$$

de x=−3 a x=3. Suponer que y se encuentra entre −10 y 30, calcular y para valores de x que difieren por 0.1.

Usamos un ciclo DO (con índice I) para los diferentes valores de x. La relación entre I y x es como sigue:

x:	−3	−2.9	−2.8	...	0	...	2.9	3
I:	1	2	3	...	31	...	60	61

Observe que hay 61 valores de x, que el eje y ocurre cuando I=31, y que x puede obtenerse de I mediante

$$x = -3.0 + FLOAT(I - 1)*0.1$$

(Ver problema 10.15.)

Suponiendo que nuestro gráfico contendrá 101 columnas, cada valor de y puede escalarse a un entero J entre 1 y 101, usando la función de escala

$$JSCALE(Y) = INT((Y + 10.0)*(100.0/40.0) + 1.5)$$

como se discutió en el problema precedente 10.16. Observe que JSCALE(0.0)=26 de modo que la columna 26 de la gráfica es el eje x.

El programa FORTRAN que grafica la parábola es como sigue: El procedimiento es el mismo como el que se discutió en la sección 10.8. En consecuencia, habrá una línea de puntos (.....) para cada eje y una cruz (X) para cada punto de la gráfica.

```
C       GRAFICA DE UNA PARABOLA
        INTEGER DOT, CROSS, BLANK, LINE(101), LINEY(101)
        DATA DOT, CROSS, BLANK, LINE, LINEY/'.', 'X', ' ', 101*' ', 101*'.'/
C       EMPIEZA NUEVA PAGINA
        WRITE(6, 10)
   10   FORMAT(1H1, 10X, 'GRAFICA DE UNA PARABOLA' ///3X, 'X', 6X, 'Y'//)
        DO 100 I = 1, 61
              X = -3.0 + FLOAT(I - 1)*0.1
              Y = 3.0*X*X - X - 8.0
              J  = INT((Y + 10.0)*(100.0/40.0) + 1.5)
              IF(I.EQ.31) GO TO 50
              LINE(26) = DOT
              LINE(J) = CROSS
              WRITE(6, 20) X, Y, LINE
   20         FORMAT(1X, 2(F4.2, 2X), 101A1)
              LINE(J) = BLANK
              GO TO 100
   50         LINEY(J) = CROSS
              WRITE(6, 20) X, Y, LINEY
  100   CONTINUE
        STOP
        END
```

10.18 Escribir un programa que lea un entero N e imprima N^2 asteriscos, de tal forma que haya N líneas con N asteriscos en una línea en las columnas 1, 3, 5,..., 2N – 1. El gráfico debe empezar en la quinta línea de una nueva página.

Primero, formamos un arreglo lineal LINE con 132 elementos (cada elemento corresponde a una columna de la página de impresión), y luego almacenamos un blanco en cada elemento de LINE. Después de leer N, almacenamos un asterisco en los elementos LINE(1), LINE(3),..., LINE(2N – 1). El programa es como sigue:

```
C       PROGRAMA QUE IMPRIME UN CUADRADO DE ASTERISCOS
        DIMENSION LINE(132)
        INTEGER AST
        DATA LINE/132*' '/, AST/'*'/
        READ(5, 10) N
   10   FORMAT(I5)
        NN = 2*N - 1
        DO 100 K = 1, NN, 2
              LINE(K) = AST
  100   CONTINUE
C       SALTE 5 LINEAS EN UNA NUEVA PAGINA
        WRITE(6, 20)
   20   FORMAT('1'/////)
C       IMPRIMIR EL ARREGLO N VECES USANDO UN CAMPO A
        DO 200 K = 1, N
              WRITE(6, 30) LINE
   30         FORMAT(1X, 132A1)
  200   CONTINUE
        STOP
        END
```

Problemas suplementarios

PROPOSICION DATA

10.19 Escribir una proposición DATA para inicializar:

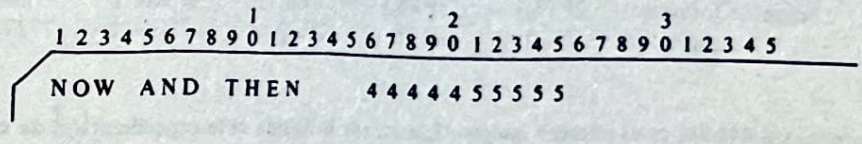
(*a*) J = 5, K = 5, L = 5, M = .TRUE. (donde M es una variable lógica)
(*b*) A(I, J)=0 para cada elemento de un arreglo A de 4×5

10.20 Encontrar la entrada:

(*a*)
```
DATA L, M, N/2*2, '2*2'/, X, Y, Z/2.5, 2*3.5/
```
(*b*)
```
DIMENSION X(6)
DATA X, Y, Z/5*2.22, 2*3.3, 4.4/
```
(*c*)
```
LOGICAL K(4)
DATA J, K, L/ 555, 3*.FALSE., .TRUE., 'TRUE'/
```

CAMPO T

10.21 Supongamos que una tarjeta de datos está perforada como sigue:



Encontrar la entrada si se ejecuta lo siguiente:

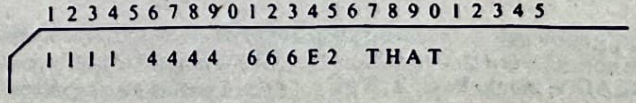
(*a*)
```
      READ(5, 10) A, B
10    FORMAT(T16, 2F5.2)
```
(*b*)
```
      READ(5, 20) X, J, K
20    FORMAT(T14, F4.2, 2I3)
```
(*c*)
```
      LOGICAL J, K
      READ(5, 30) I, J, K, C, D
30    FORMAT(T16, I3, T4, 2L4, T16, 2F3.1)
```

10.22 Encontrar la salida para cada respuesta de programa:

(*a*)
```
J = 1111
K = 5555
L = 8888
WRITE(5, 10) J, K, L
10  FORMAT(T16, I5, T6, I5, T26, I5)
```
(*b*)
```
A = 22.22
B = 77.77
C = 11.11
WRITE(5, 20) A, B, C
20  FORMAT(T16, F5.1, 5X, F5.1, T1, F5.1)
```

CAMPO G

10.23 Supongamos que una tarjeta de datos está perforada como sigue:



Encontrar la entrada si se ejecuta lo siguiente:

(*a*)
```
      LOGICAL J, K
      READ(5, 10) N, X, Y, J, K
10    FORMAT(5G5.2)
```
(*b*)
```
      LOGICAL J, K
      READ(5, 20) X, N, Y, J, K
20    FORMAT(G7.1, 4G4.2)
```

10.24 Encontrar la salida si los valores de N, J, K, X, Y en el problema 10.23 se imprimen utilizando

```
      WRITE(6, 30) N, J, K, X, Y
30    FORMAT(1X, 5G10.3)
```

10.25 Supongamos que un arreglo lineal A con 5 elementos contiene respectivamente

$$44.444,\ 0.66666 \times 10^{-6},\ 33.3 \times 10^{2},\ 22.2 \times 10^{8},\ 333.3 \times 10^{-2}$$

Encontrar la salida si la instrucción es

```
      DO 100 K = 1, 5
            WRITE(6, 50) A(K)
50          FORMAT(11X, G12.4)
100   CONTINUE
```

FACTOR DE ESCALA

10.26 Encontrar el valor interno dado el siguiente código de formato y el número externo como entrada:

	(*a*)	(*b*)	(*c*)	(*d*)	(*e*)
Código de formato:	2PF5.1	−3PF5.1	3PE5.1	−4PE10.1	−1PE10.1
Entrada:	77.77	77.77	77.77	77.77E + 2	7777E + 2

10.27 Supongamos que 444.888 es el número interno. Encontrar la salida si la especificación de campo es

(*a*) 3PF10.2 (*c*) −PF10.3 (*e*) −1PE15.5
(*b*) −2PF10.1 (*d*) 4PE15.3 (*f*) 2PE15.4

10.28 Supongamos que X contiene 444.777. Encontrar la salida si la instrucción es

```
      WRITE(6, 10) X, X, X, X, X, X
10    FORMAT(1X, F12.2, 3X, E12.4, 3X, 2PF12.2, 3X, E12.4, 3X, 0PF12.2, 3X, E12.4)
```

REGLA DEL PARENTESIS IZQUIERDO

10.29 Considere el segmento de programa

```
      DIMENSION K(6)
      DATA K/4*33, 2*66/
      WRITE(6, 10) K
```

Encontrar la salida si la proposición FORMAT es

(*a*) 10 FORMAT(1X, I2, (I4, I6)) (*b*) 10 FORMAT(1X, I2, 2(3X, I2), 8X, I2)

10.30 Considere la proposición

```
      READ(5, 20) A, B, C, X, Y, Z
```

Escriba una proposición FORMAT tal que

A se lee de la primera tarjeta utilizando F10.2;

B, C, y X se leen de la segunda tarjeta utilizando F10.3, F10.2 y F10.4;

Y, Z se leen de la tercera tarjeta con F10.3 y F10.2.

FORMATO A TIEMPO DE EJECUCION

10.31 Suponiendo que la capacidad de caracteres es M=4, describa la entrada

```
INTEGER FORM(5)
DATA FORM/'(1X,', 'I3,', '3X,', 'F6', '.3)'/
```

10.32 Supongamos que J y X son arreglos lineales con 5 elementos cada uno y que tienen los siguientes valores

333, 33333, 33, 333333, 3 y 4.4, 66666.66, 1.1, 88.88888, 2222.2

respectivamente. Encontrar la salida, si se ejecuta lo siguiente, usando el arreglo FORM del problema 10.31:

```
      DATA IA, IB/'I7,', 'F10'/
      DO 100 K = 1, 5
            IF(J(K).GE.1000) FORM(2) = IA
            IF(X(K).GE.100.0) FORM(4) = IB
            WRITE(6, FORM) J(K), X(K)
100   CONTINUE
```

10.33 Supongamos que el arreglo lineal J contiene 75 enteros positivos menores que 10,000. Escribir un segmento de programa que imprima los números en una columna, de tal forma que estén justificados a la izquierda, por ejemplo,

```
28
111
5
3344
62
⋮
```

GRAFICOS

10.34 Escribir un programa que grafique la función $y=\text{sen } x$ para $-10 \leq x \leq 10$. (Recordemos $-1 \leq y \leq 1$ para cualquier valor de x.)

10.35 Supongamos que los valores $y_1, y_2, \ldots, y_N$ están almacenados en un arreglo lineal Y con N elementos. Escribir un programa SUBROUTINE

```
ENDS(Y, N, YMIN, YMAX)
```

que encuentre el valor mínimo YMIN y el valor máximo YMAX, de los elementos del arreglo Y.

10.36 Escribir un programa que grafique la función

$$y = x^3 + 2x^2 - 15x - 8$$

desde $x=-5$ a $x=4$, donde los valores de x difieren en 0.2. En particular, almacenar primero todos los valores de y en un arreglo lineal y luego, usar la subrutina ENDS del problema 10.35 y una función de escala similar a la del problema 10.16, para escalar y a un entero J entre 1 y 101.

10.37 Escribir un programa que acepte 2 enteros positivos J y K e imprima un rectángulo de J•K asteriscos, tal que haya J líneas con K asteriscos que aparezcan en las columnas $1, 3, 5, \ldots, 2K-1$. (La gráfica debe empezar en la tercera línea de una nueva página.)

Respuestas a problemas suplementarios seleccionados

10.19 (*a*) DATA J, K, L, M/3*5, .TRUE./ (*b*) DATA A/20*0.0/

10.20 (*a*) L y M contienen 2, N contiene los cuatro caracteres $2*2_b$ (suponiendo M=4), X contiene 2.5; Y y Z contienen 3.5.
(*b*) X(1),..., X(5) contienen 2.22, X(6) y Y contienen 3.3 y Z contiene 4.4.
(*c*) J contiene 555; K(1), K(2) y K(3) contienen .FALSE.; K(4) contiene .TRUE.; L contiene la cadena de caracteres TRUE.

10.21 (*a*) A contiene 444.44 y B contiene 555.55.
(*b*) X contiene 0.44, J contiene 444 y K contiene 555.
(*c*) I contiene 444; J contiene .FALSE.; K contiene .TRUE.; C contiene 44.4 y D contiene 44.5.

10.22 (*a*) $_{bbbbbb}5555_{bbbbbb}1111_{bbbbbb}8888$ (*b*) $11.1_{bbbbbbbbbb}22.2_{bbbbbb}77.8$

10.23 (*a*) 11110, 444.40, 6.66E2(ó : 666.0), .TRUE., .FALSE.
(*b*) 111104.4, 4406, 0.66E2(ó : 66.0), .TRUE., .TRUE.

10.24 Justificado a la derecha en sus campos:

(*a*) 11110, T, F, $444._{bbbb}$, $666._{bbbb}$ (*b*) 4406, T, T, $0.111E_b06$, 66.0_{bbbb}

10.25 Ver figura 10-5.

Fig. 10-5

10.26 0.7777; 77770.; 0.07777; 77.77E+2, es decir, 7777.0 (no se afecta); 777.7E+2, es decir, 77770.0 (no se afecta).

10.27 (*a*) 444888.00 (*c*) 44.489 (*e*) 0.044489E 04
(*b*) 4.4 (*d*) 4450.E − 01 (*f*) 44.49E 01

10.28 $_{bbbbbb}444.78_{bbbbb}0.4448E_b03_{bbbbbbb}44477.70_{bbbbbb}44.48E_b01_{bbbbbbbbb}444.78_{bbbbb}0.4448E_b03$

10.29 (*a*) $33_{bb}33_{bbbb}33$ en una línea, $_b33_{bbbb}66$ en la siguiente línea, y $_b66$ en la tercera línea.
(*b*) $33_{bbb}33_{bbb}33_{bbbbbbbb}33$ en una línea, y $_{bb}66_{bbb}66$ en la siguiente línea.

10.30 20 FORMAT(F10.2/(F10.3, F10.2, F10.4/))

10.31 El arreglo FORM es almacenado como sigue

FORM(1)	(	1	X	,
FORM(2)	I	3	,	
FORM(3)	3	X	,	
FORM(4)	F	6		
FORM(5)	.	3	)	

10.32 Ver figura 10-6.

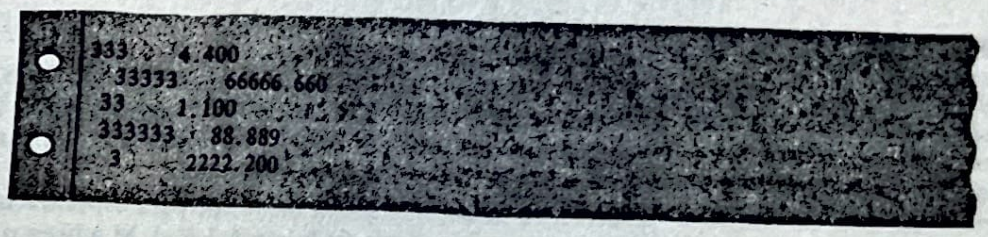


Fig. 10-6

10.33 Sugerencia: Usar formato a tiempo de ejecución con códigos de formato I1, I2, I3 e I4, según que J contenga uno, dos, tres o cuatro dígitos.

Capítulo 11

Tópicos diversos del Fortran

11.1 INTRODUCCION

Este capítulo trata varios aspectos adicionales del lenguaje de programación Fortran. En particular, consideramos las proposiciones de declaración de tipo en forma general, la proposición IMPLICIT, la proposición DOUBLE PRECISION, las variables COMPLEJAS y las proposiciones COMMON y EQUIVALENCE.

11.2 PROPOSICIONES DE DECLARACION DE TIPO

Hay varios tipos de variables: REALES, ENTERAS, LOGICAS, de DOBLE PRECISION y COMPLEJAS. Las variables REALES y ENTERAS se discutieron en el capítulo 2, las variables LOGICAS se discutieron en el capítulo 9 y las variables COMPLEJAS y de DOBLE PRECISION se discutirán en las secciones 11.4 y 11.5.

Fortran nos permite utilizar la "primera letra" como una convención de tipo para especificar el tipo de una variable. O sea que cualquier variable cuyo nombre comience con la letra I, J, K, L M o N es una variable entera y, cualquier otra, es una variable real. Sin embargo, esta convención se puede romper utilizando una proposición de declaración de tipo.

Una proposición de declaración de tipo tiene la forma

TIPO *lista de variables*

donde los nombres de variables están separadas por comas. La proposición de declaración de tipo implica que las variables que están en la lista deben ser del tipo dado. Por ejemplo,

```
REAL DINERO, TASA, NUEVO
INTEGER X, Y, NUEVO
LOGICAL A, B
DOUBLE PRECISION S, T
COMPLEX RAIZ, COEF
```

declara DINERO, TASA y NUEVO como variables reales; X, Y y NUEVO como variables enteras y así, sucesivamente. Obsérvese que TASA pudo no haberse incluido en la primera proposición o NUEVO en la segunda. En efecto, algunos programadores consideran que vale la pena listar cualquier variable en una proposición de declaración de tipo, haciendo caso omiso de su primera letra.

Los nombres de arreglos también pueden aparecer en proposiciones de declaración de tipo. En efecto, se puede evitar el uso de una proposición DIMENSION separada, al incluir el tamaño del arreglo en la proposición de declaración de tipo. Por ejemplo,

```
INTEGER COUNT(30)
REAL NUMBER(4, 5), LAST
```

declara a COUNT como un arreglo lineal entero de 30 elementos y a NUMBER, como un arreglo real de 4×5.

Las proposiciones de declaración de tipo no son ejecutables; deben aparecer al comienzo del programa antes de la proposición DATA y antes de que se utilice cualquier variable.

11.3 PROPOSICION IMPLICIT

La convención de tipo implícita a partir de la "primera letra" puede ampliarse con el uso de la proposición IMPLICIT. Ilustramos esto con un ejemplo. Consideremos la proposición

```
IMPLICIT INTEGER(B, D, W – Z), LOGICAL(A, N – P), COMPLEX(F – H)
```

Esta le informa al compilador que, a menos que se establezca explícitamente por una proposición de declaración de tipo, todos los nombres de variables que comiencen por las letras B, D, W, X, Y, Z son variables enteras, las que comiencen con las letras A, N, O y P son variables lógicas y las que se comiencen por las letras F, G y H son variables complejas. Obsérvese que un intervalo de letras se indica escribiendo la primera y última letras, separadas por un signo menos. Nótese, además, que las letras y su intervalo van después del TIPO y que ellas van encerradas entre paréntesis y separadas por comas.

En la utilización de la proposición IMPLICIT, notamos lo siguiente.

1. Usualmente sólo una proposición IMPLICIT puede aparecer en cualquier programa o subprograma y debe ser la primera proposición posible. Esto es, debe ser la primera proposición en un programa principal.
2. Se puede anular una proposición IMPLICIT con una proposición de declaración de tipo. Además, la acostumbrada convención de tipo a partir de la "primera letra" se mantiene para cualquier variable que no esté incluida en la proposición IMPLICIT o en proposiciones de declaración de tipo. Por ejemplo, supongamos que un programa tiene las proposiciones

```
IMPLICIT DOUBLE PRECISION (A – D, S), INTEGER(F – G)
COMPLEX ALPHA, BETA
INTEGER SET, CLASS
```

   Entonces, ALPHA y BETA son variables COMPLEJAS y SET y CLASE son variables enteras; teniendo en cuenta las proposiciones de declaración de tipo ANSWER y SUMA son variables de DOBLE PRECISION, según la proposición IMPLICIT. Por otra parte, como TASA Y KONT no están incluidas en las proposiciones dadas, entonces son variables REAL y ENTERA respectivamente.
3. Los parámetros de un subprograma, así como el nombre de un subprograma FUNCTION, están cobijados por una proposición IMPLICIT que ya se haya incluido en el subprograma.

11.4 DOBLE PRECISION

Normalmente un computador puede almacenar siete u ocho dígitos significativos en cualquier localización de memoria y su aritmética (llamada *aritmética de precisión sencilla*), retendrá un número similar de dígitos significativos. Sin embargo, Fortran nos permite obtener resultados con mayor precisión, utilizando la llamada *aritmética y valores de doble precisión.* En este caso, cada valor se almacena en dos (2) localizaciones de memoria; de manera que el número de dígitos significativos aproximadamente se duplica. (Por ejemplo, las series IBM 360/370 retienen siete dígitos significativos en la aritmética ordinaria, pero retienen 16 dígitos significativos en la aritmética de doble precisión.) En esta sección, discutiremos las reglas concernientes a la aritmética de doble precisión.

Nota. Puesto que los valores de doble precisión ocupan dobles localizaciones de memoria y puesto que su aritmética puede tomar 2 a 10 veces más tiempo de ejecución, la doble precisión solamente deberá utilizarse cuando se requiera extremada precisión.

Variables

Los arreglos y las variables de doble precisión se declaran utilizando la proposición DOUBLE PRECISION, mencionada en la sección 11.2.

Constantes

Las constantes de doble precisión se escriben de modo similar a las constantes reales de precisión sencilla. En particular, se debe colocar un *punto decimal* en cualquier constante de doble precisión.

1. *Sin exponente.* Una constante real escrita sin un exponente se interpreta como una constante de precisión sencilla, a menos que el número de dígitos significativos exceda al máximo número permisible para una constante real de precisión sencilla. Por ejemplo, para las máquinas IBM 360/370, que permiten siete dígitos significativos, la constante 123.456 se tomará como de precisión sencilla, mientras que 123.456000000 se tomará como de doble precisión.
2. *Con exponente.* Las constantes de precisión sencilla se representan en forma exponencial, utilizando el caracter E; por ejemplo, 46.8086E7 es una constante de precisión sencilla. Las constantes de doble precisión se representan en forma exponencial, utilizando la letra D en lugar de la letra E. De esta manera, 46.8086D7 representa el mismo número en doble precisión; sin embargo, se almacena internamente, con un total de 16 dígitos significativos, como

$$0.4680860000000000 \times 10^{9}$$

(Esto es, se añade una cadena de ceros hasta obtener los 16 números de dígitos significativos.)

Advertimos que no hay casos tales, como enteros de doble precisión.

Entrada/Salida

El campo D se utiliza para la entrada/salida de valores de doble precisión. Aparte del hecho de que los campos D permiten utilizar mayor número de dígitos decimales, el uso del campo D es completamente similar al uso del campo E; por ejemplo, la especificación de campo tiene la forma

Dw.d

donde *w* es el tamaño del campo y *d* es el número de dígitos decimales. Por esta razón, se aconseja a los lectores revisar el capítulo 3 sobre campos E, con el fin de comprender esta discusión.

Operaciones aritméticas

La aritmética de doble precisión tiene lugar cuando ambos operandos son de doble precisión. Muchos compiladores permiten la mezcla de términos de precisión sencilla y doble precisión. En tal caso, el término de precisión sencilla se convierte a doble precisión (añadiendo una cadena de 0's) y entonces la aritmética de doble precisión se lleva a cabo. Por otra parte, muchos compiladores no permiten la mezcla de tipos entero y de doble precisión, excepto en el caso de la exponenciación, en que los valores de doble precisión siempre pueden elevarse a un exponente entero. Observemos también que un valor de doble precisión con signo negativo no puede elevarse a una potencia real o de doble precisión, como fue el caso de los números reales negativos. (Véase página 28.)

Proposiciones de asignación

Como en el caso de la precisión sencilla, los tipos de los dos lados del signo = pueden ser diferentes. El valor que se va a almacenar se convierte siempre al tipo de la localización de almacenamiento. Además, los valores de doble precisión se truncan, no se redondean cuando se convierten a precisión sencilla. Consideremos, por ejemplo, el segmento

```
DOUBLE PRECISION A
A = 1.0
B = 123.456789098
J = 123.456789098
K = 3.4444D3
```

Entonces,

1. La constante se convierte a doble precisión y se almacena en A.
2. Solamente se almacena 123.4567 en B; esto es, la constante se trunca después de los siete dígitos significativos.

3. Solamente se almacena 123 en J; esto es, la constante se trunca en el punto decimal.
4. Aunque 3.4444D3 es igual a 3444.4, solamente el valor 3 se asigna a K, puesto que la constante se trunca en el punto decimal,

Funciones de biblioteca

Las funciones de biblioteca DFLOAT y DBLE convierten, respectivamente, valores enteros y de precisión sencilla, a doble precisión; y la función SNGL convierte un valor de doble precisión, a precisión sencilla. Estas también son funciones de biblioteca que suministran cálculos en doble precisión para muchas de las funciones frecuentemente utilizadas. Por ejemplo, si DX es un argumento de doble precisión, entonces

```
DSQRT(DX)
```

dará la raíz cuadrada de DX, utilizando aritmética de doble precisión. Análogamente, DABS(DX), DLOG(DX), DSIN(DX) dan el valor absoluto, el logaritmo y el seno de DX, en doble precisión. Generalmente la letra D en frente de una función denota cálculos de doble precisión. En el apéndice A se da una lista de muchas otras funciones. Advertimos que tendríamos que utilizar

```
DLOG(DBLE(X))
```

para obtener el logaritmo natural en doble precisión de un valor real X de precisión sencilla, puesto que el argumento de la función DLOG tiene que ser de doble precisión.

Subprogramas

Los valores de doble precisión pueden transmitirse a y de un subprograma por la vía de los argumentos (lo mismo que cualquier otro tipo de valores); sin embargo, los nombres de estos argumentos y los parámetros correspondientes deben declararse como de doble precisión, tanto en el llamado del programa, como en el subprograma. Un subprograma FUNCTION, en sí mismo, puede declararse como de doble precisión, escribiendo DOUBLE PRECISION al comienzo de la proposición que define la función, declarando el NOMBRE como de doble precisión en el subprograma. Por ejemplo, los siguientes subprogramas son equivalentes:

```
DOUBLE PRECISION FUNCTION SUM(A, B)
DOUBLE PRECISION A, B
SUM = A + B
RETURN
END
```

y

```
FUNCTION SUM(A, B)
DOUBLE PRECISION SUM, A, B
SUM = A + B
RETURN
END
```

Más aún, muchos compiladores requieren que el NOMBRE en doble precisión de un subprograma FUNCTION, sea declarado en el llamado de programa. Esto es, la proposición

```
DOUBLE PRECISION SUM
```

o su equivalente, deben aparecer en el llamado de programa, si se utiliza el subprograma SUM. (Este requerimiento no es universal, de manera que un programador deberá averiguar los requerimientos exactos para el compilador particular que se utilice.) Por otro lado, no hay cosas tales como un subprograma DOUBLE PRECISION SUBROUTINE, puesto que ningún valor se asigna al NOMBRE de la subrutina.

EJEMPLO 11.1

Suponiendo que se utiliza una máquina IBM 360/370, el segmento de programa

```
      DOUBLE PRECISION DA, DB
      DA = 2./3.
      DB = 2.0D0/3.0D0
      WRITE(6, 10) DA, DA
      WRITE(6, 10) DB, DB
10    FORMAT(1X, D23.16, 10X, D12.4)
```

producirá

```
0.6666666000000000D 00          0.6667D 00
0.6666666666666666D 00          0.6667D 00
```

La diferencia ocurre porque 2./3. se calcula en precisión sencilla y se trunca después de siete dígitos significativos, es decir, tiene el valor 0.6666666. Cuando 2./3. se almacena en DA, se convierte en doble precisión al añadírsele una cadena de 0's. Por otra parte, 2.0D0/3.0D0 se calcula en doble precisión y se trunca después de 16 dígitos significativos, es decir, tiene el valor 0.6666666666666666. (No hay redondeo en estos cálculos.) Sin embargo, cuando se imprimen estos valores, se redondean en los cuatro dígitos significativos, como se especifica en los códigos de formato.

11.5 NUMEROS COMPLEJOS

Los compiladores Fortran suministran las operaciones aritméticas para manejar los números complejos. Revisaremos primero la aritmética de los números complejos y luego discutiremos la manera de manipularlos en Fortran.

Un número complejo z usualmente se escribe en la forma

$$z = a + bi$$

donde a, b son números reales ordinarios e $i = \sqrt{-1}$. La parte a se denomina *parte real* de z y la parte b se llama la *parte imaginaria* de z. (Un número real a puede identificarse como el número complejo $a+0i$.)

La suma y producto de los números complejos pueden obtenerse utilizando las leyes usuales de conmutación y distribución del álgebra con $i^2 = -1$:

$$(a + bi) + (c + di) = a + c + bi + di = (a + c) + (b + d)i$$
$$(a + bi)(c + di) = ac + bci + adi + bdi^2 = (ac - bd) + (bc + ad)i$$

El *conjugado* del número complejo $z=a+bi$ se denota y define como

$$\bar{z} = a - bi$$

(Nótese que $z\bar{z}=a^2+b^2$.) Si $z\neq 0$, es decir, si $a\neq 0$ ó $b\neq 0$, entonces el inverso z^{-1} de z y la división por z están dados como

$$z^{-1} = \frac{\bar{z}}{z\bar{z}} = \frac{a}{a^2 + b^2} + \frac{-b}{a^2 + b^2}i \qquad \text{y} \qquad \frac{w}{z} = wz^{-1}$$

donde w es también un número complejo. Definimos también

$$-z = -1z \qquad \text{y} \qquad w - z = w + (-z)$$

El *valor absoluto* o *magnitud* de $z=a+bi$, se escribe como $|z|$, y se define como

$$|z| = \sqrt{a^2 + b^2} = \sqrt{z\bar{z}}$$

EJEMPLO 11.2

Supongamos que $z=2+3i$ y $w=5-2i$; entonces

$$z + w = (2 + 3i) + (5 - 2i) = 2 + 5 + 3i - 2i = 7 + i$$
$$zw = (2 + 3i)(5 - 2i) = 10 + 15i - 4i - 6i^2 = 16 + 11i$$
$$\bar{z} = \overline{2 + 3i} = 2 - 3i \quad \text{y} \quad \bar{w} = \overline{5 - 2i} = 5 + 2i$$
$$\frac{w}{z} = \frac{5 - 2i}{2 + 3i} = \frac{(5 - 2i)(2 - 3i)}{(2 + 3i)(2 - 3i)} = \frac{4 - 19i}{13} = \frac{4}{13} - \frac{19}{13}i$$
$$|z| = \sqrt{4 + 9} = \sqrt{13} \quad \text{y} \quad |w| = \sqrt{25 + 4} = \sqrt{29}$$

Puesto que un número complejo consta de dos partes, la parte real y la parte imaginaria, se requieren dos localizaciones de memoria para almacenar un número complejo. Las reglas concernientes a las constantes y variables complejas, así como su aritmética, son las siguientes.

Variables

Las variables y arreglos se declaran utilizando la proposición COMPLEX, mencionada en la sección 11.2.

Constantes

Fortran representa una constante compleja por una pareja de constantes *reales*, separadas por una coma y encerradas entre paréntesis. El primer número real denota la parte real del número complejo y el segundo, la parte imaginaria. Cada uno de los dos números puede escribirse como una constante real, con o sin exponente, pero nunca pueden escribirse como constantes enteras. Así que,

$$1 + 2i$$

puede escribirse en Fortran de alguna de las siguientes maneras:

```
(1.0, 2.0)
(0.1E1, 2.0)
(1.0, 2.0E0)
(0.1E1, 0.2E1)
```

Pero no puede representarse por

```
(1, 2)
```

Por otra parte, una pareja ordenada de variables reales o de expresiones no se acepta para la definición de un valor complejo. Por ejemplo, dado que X es real, no podemos utilizar

```
(X, 2.0)
```

para definir un valor complejo. Sin embargo, Fortran nos proporciona una función de biblioteca tipo complejo CMPLX que convierte una pareja ordenada de *expresiones reales*, en su valor constante complejo. Así que,

```
CMPLX(X, 2.0)    y    CMPLX(X + 2.0, X - 2.0)
```

son las expresiones Fortran que nos permiten escribir los números complejos $x+2i$ y $(x+2)+(x-2)i$.

Entrada/Salida

No hay un campo especial para la entrada/salida de números complejos. Sin embargo, puesto que dos números reales corresponderán a cada número complejo, se deben dar dos campos de especificaciones para cada variable compleja. Por ejemplo,

```
      COMPLEX Z
      READ(5, 10) Z
10    FORMAT(F10.2, 10X, F10.2)
```

asignará el valor en las columnas 1-10 a la parte real de z y el valor en las columnas 21-30 a la parte imaginaria de z. Por lo tanto, si la tarjeta de datos se prepara como

```
          1         2         3
12345678901234567890123456789012345
    12.34     333.33    13.579
```

entonces Z tendrá el valor $12.34+13.579i$. Naturalmente, los campos de especificación pueden ser campo E, campo F o Campo G, o una combinación de ellos.

Operaciones aritméticas

La aritmética compleja tiene lugar cuando ambos operandos son complejos, o cuando el uno es real y el otro es complejo. En el último caso, la variable real, o sea a, se convierte a su equivalente complejo $(a, 0.0)$ para que la aritmética compleja pueda llevarse a cabo. Por ejemplo si X es real y C es compleja, entonces

```
X + (1.0, 2.0),    X + C,    C - X,    X/C,    C*X,    C + 1.0
```

son expresiones Fortran aceptables. En caso de que el compilador acepte la mezcla de reales y enteros, los valores enteros pueden ser usados también con los números complejos; por ejemplo C+1 es aceptable y se hará la aritmética en números complejos. Sin embargo, hay una restricción especial que se mantiene para los números complejos, pero no para los reales; un número complejo no puede elevarse a una potencial real. En otras palabras, un número complejo solamente puede elevarse a una potencia entera.

Proposiciones de asignación

Valores reales, enteros y complejos pueden asignarse a variables complejas. Por ejemplo, si A y B son variables complejas y X y Y son variables reales, entonces las siguientes proposiciones de asignación son válidas:

```
A = (1.0, 0.2E1)
A = A + B**2
A = 25
A = 3*X
A = 3*B + Y - 8.0
A = CMPLX(X, Y + 2.0)
```

Específicamente, los valores reales y enteros se convierten a valores complejos, antes de que sean almacenados en la variable compleja A. Por otro lado, los valores complejos no pueden asignarse a variables reales o enteras. Así, por ejemplo,

```
X = A + B
K = B**2
```

no son válidas.

Funciones de biblioteca tipo complejo

La función CMPLX(X, Y) discutida anteriormente toma dos argumentos reales X y Y y los convierte en un número complejo $X+Yi$. Las partes real e imaginaria y la magnitud (o valor absoluto) de un número complejo se pueden obtener a partir de las funciones REAL, AIMAG y CABS, respectivamente. Esto es, si $C=X+Yi$, entonces

```
REAL(C)    dará el valor X
AIMAG(C)   dará el valor Y
CABS(C)    dará el valor √(X**2 + Y**2)
```

Hay otras funciones de biblioteca tipo complejo que son análogas a las funciones reales. Por ejemplo, si C es una variable o expresión compleja, entonces

```
CSQRT(C)
```

dará la raíz cuadrada compleja de C. De modo análogo, CSIN(C), CCOS(C), CLOG(C) y CEXP(C) evalúa el seno, coseno, logaritmo y exponencial complejos, respectivamente, de C. Generalmente, la letra C en frente de una función denota una función compleja. En el apéndice A se da una lista de muchas de tales funciones.

Subprogramas

Los valores complejos se utilizan en los subprogramas, de la misma manera que se utilizan los valores en doble precisión. Específicamente, los valores complejos se transmiten a y de un subprograma por la vía de los argumentos; los nombres de estos argumentos y los parámetros correspondientes deben declararse como complejos, tanto en el llamado de programa como en el subprograma. Además, un subprograma FUNCTION puede declararse como complejo escribiendo COMPLEX al comienzo de la proposición que define la función o declarando el NOMBRE como complejo en el subprograma. Por ejemplo, las siguientes proposiciones son equivalentes:

```
COMPLEX FUNCTION XYZ(A, B, C)        ó    FUNCTION XYZ(A, B, C)
COMPLEX A, B                              COMPLEX XYZ, A, B
```

Cada una declara que XYZ es una función de valor complejo y que A y B son variables complejas. Además, como en el caso de las funciones de doble precisión, muchos compiladores requieren que el NOMBRE de un subprograma FUNCTION tipo complejo, sea declarado en el llamado de programa. Esto es, la proposicion

```
COMPLEX XYZ
```

deberá aparecer en el llamado de programa que utilice el anterior subprograma complejo XYZ.

11.6 PROPOSICION GO TO ASIGNADA, PROPOSICION ASSIGN

Algunos compiladores proporcionan otra proposición de transferencia de control, denominada la proposición GO TO asignada. La proposición GO TO asignada y una proposición asociada llamada proposición ASSIGN, tienen la siguiente forma:

```
ASSIGN n TO J
      ⋮
GO TO J, (n1, n2, . . . , nk)
```

Donde, n, n_1, n_2, . . ., n_k son constantes enteras sin signo, utilizadas para establecer números de proposición (rótulos) y J es una variable entera. La proposición ASSIGN asigna el valor de la constante n a la variable J. Cuando se ejecuta la proposición GO TO asignada, el control se transfiere a la proposición rotulada como n, o sea, a aquella cuyo valor es el valor actual de J.

Recapitulamos las reglas para las proposiciones ASSIGN y GO TO asignadas, como sigue:

1. La variable J debe ser una variable entera (no subindicada).
2. La variable J no puede ser cambiada excepto por otra proposición ASSING. Por ejemplo, el segmento

```
ASSIGN 25 to LAST
LAST = LAST + 5
GO TO LAST, (20, 25, 30, 35)
```

   no es válido.
3. El valor de J debe aparecer en la lista de números de proposiciones que siguen a J, en la proposición GO TO asignada.

11.7 ENTRADAS MULTIPLES Y RETORNOS A UN SUBPROGRAMA

(*a*) Entrada múltiple

Normalmente, cuando un nombre de subprograma (FUNCTION o SUBROUTINE) aparece en un llamado de programa, la entrada al subprograma se hace por la primera proposición ejecutable que sigue a la proposición que define el subprograma. Algunos compiladores proporcionan instrucciones para entrar a un subprograma por diferentes puntos de entrada. Para indicar puntos de entrada alternativos, se utiliza la proposición ENTRY en el subprograma. Esta proposición tiene la forma

ENTRY NOMBRE (lista de parámetros)

donde el NOMBRE del punto de entrada es diferente del nombre del subprograma. La proposición ENTRY no es ejecutable y se coloca en el subprograma en el punto donde debe realizarse la entrada. La proposición no tendrá efecto sobre la lógica del subprograma.

Se podrían tener varias declaraciones ENTRY en un subprograma, de manera que cada una definiera un punto de entrada diferente. Evidentemente, cada punto de entrada debe tener su propio nombre. Además, los parámetros asociados con un punto de entrada no necesariamente deben corresponder con los parámetros del subprograma, ni tampoco deben corresponder con los parámetros de otros puntos de entrada. Sin embargo, cuando un punto de entrada particular NOMBRE se requiera en el llamado de programa, la lista de argumentos de la proposición de llamado debe corresponder con los parámetros de la proposición ENTRY.

La forma de la proposición de llamado para un punto de entrada a un subprograma se hace análogamente a la de los subprogramas FUNCTION o SUBROUTINE. En ambos casos, cuando se usa la entrada a NOMBRE mediante un llamado de programa, la ejecución empezará en la primera proposición ejecutable que sigue a la declaración ENTRY en el subprograma.

El siguiente esquelto de programa ilustra los conceptos anteriores:

Programa principal

```
.....
CALL ABC(P, Q, R)
.....
CALL TWOA(E, F)
.....
CALL ONEA(S, M)
.....
STOP
END
```

Subprograma

```
      SUBROUTINE ABC(X, Y, Z)
      .....
      GO TO 10
      ENTRY ONEA(W, K)
      .....
      GO TO 10
      ENTRY TWOA(U, V)
      .....
10    RETURN
      END
```

(*b*) Retorno múltiple (solamente para SUBROUTINE)

Normalmente, cuando se encuentra una proposición RETURN en una SUBRUTINA, se transfiere el control a la primera proposición ejecutable que sigue a la proposición CALL en el llamado de programa. Fortran también nos permite retornar a diferentes puntos del llamado de programa. Ilustramos esta facilidad, utilizando el siguiente esqueleto de programa.

Programa principal

```
DIMENSION TEST(100)
.....
CALL GRADE(A, 30, B, C, 20, TEST)
.....
END
```

Subrutina

```
      SUBROUTINE GRADE(R, *, S, T, *, W)
      DIMENSION W(100)
      .....
      RETURN 2
      .....
      RETURN
      .....
      RETURN 1
      END
```

Obsérvese que hay dos asteriscos en la lista de parámetros de la proposición que define la SUBRUTINA. Obsérvese también que las proposiciones rotuladas como 30 y 20 en la lista de argumentos corresponden, respectivamente, a esos dos asteriscos. En consecuencia, las proposiciones RETURN 1 y RETURN 2 corresponderán a las proposiciones rotuladas como 30 y 20, respectivamente, en la forma siguiente. Si se encuentra RETURN 1 en el subprograma, entonces se transfiere el control a la proposición numerada como 30 en el llamado de programa y si se encuentra RETURN 2, se transfiere el control a la proposición rotulada como 20 en el llamado de programa. Por otra parte, cuando se encuentra RETURN, se devuelve el control a la primera proposición ejecutable que sigue a la proposición CALL.

Generalmente, la proposición RETURN múltiple tiene la forma

```
RETURN n
```

donde n es una constante entera sin signo. Esta proposición puede aparecer solamente en una subrutina que tenga al menos n asteriscos en su lista de parámetros y donde cada asterisco corresponda a un número de proposición, en la lista de argumentos. Entonces, cuando se encuentre RETURN n en la subrutina, se transfiere el control a la proposición en el llamado de programa, cuyo rótulo esté como el n-ésimo rótulo en la lista de argumentos.

11.8 PROPOSICIONES COMMON NO ROTULADAS

Recordemos que en un subprograma las variables, diferentes de los parámetros, son locales para ese subprograma. En consecuencia, el único camino para pasar información entre una llamada de subprograma y un subprograma es especificarlas en la lista de argumentos de los programas. Sin embargo, esta forma puede ser complicada si un gran número de localizaciones de memoria son compartidas por varios subprogramas. Fortran nos provee con una forma alternada para que los subprogramas puedan compartir espacios comunes de memoria. Específicamente, se puede definir un área de almacenamiento común utilizando una proposición COMMON y así los subprogramas, mediante declaraciones y descripciones apropiadas, pueden tener acceso a cualquier información almacenada en esa área.

Hay dos clases de proposiciones COMMON: COMMON rotulado y COMMON no rotulado (blanco). Ambas formas son no ejecutables y deben estar colocadas al principio de cualquier programa, antes de cualquier proposición ejecutable. En esta sección, discutiremos las proposiciones COMMON no rotuladas y en la sección siguiente, discutiremos las proposiciones COMMON rotuladas.

Una proposición típica de COMMON no rotulado es como sigue:

```
DIMENSION Y(2, 3), W(2)
COMMON X, Y, Z, W
```

Observe que la proposición COMMON empieza con la palabra COMMON, seguida de una lista de variables, incluyendo nombres de arreglos, separadas por comas. Se puede obviar el uso de proposiciones DIMENSION separadas, para especificar los arreglos en una proposición COMMON. Por ejemplo, cada una de las siguientes formas,

```
COMMON X, Y(2, 3), Z, W(2)     ó     DIMENSION Y(2, 3)
                                     COMMON X, Y, Z, W(2)
```

es equivalente a la declaración anterior.

Uno cualquiera de los COMMON anteriores agrupa un bloque común de 10 localizaciones de memoria (1+6+1+2), donde las posiciones de memoria están localizadas de acuerdo con el orden de la lista de variables. En consecuencia, en nuestro ejemplo las localizaciones de memoria son denominadas como en la figura 11-1.

Ilustraremos ahora cómo se usa la proposición COMMON y cómo se comparte el área de almacenamiento común, mediante el esqueleto de programa dado en la figura 11-2. (Para mayor claridad, los subprogramas emplean exactamente la misma proposición COMMON, aunque esto no es necesario. Ver *nota A* en la página 275.) Obsérvese que el programa principal define un área de almacenamiento común de 20 localizaciones y llama 3 subrutinas y un subprograma FUNCTION. Las subrutinas XXX, YYY y la

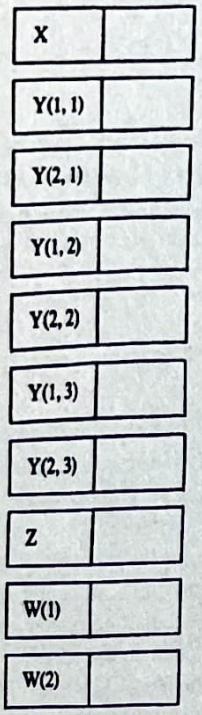


Fig. 11-1

```
Programa principal
COMMON A(10), B, X(2, 4), Q
.....
CALL XXX(S, I)
.....
CALL WWW(P1, P2, P3)
.....
T = TAX + ZZZ(U, V)
.....
CALL YYY
.....
END

Subprograma 1
SUBROUTINE XXX(F, K)
COMMON A(10), B, X(2, 4), Q
.....
RETURN
END

Subprograma 2
SUBROUTINE YYY
COMMON A(10), B, X(2, 4), Q
.....
RETURN
END

Subprograma 3
FUNCTION ZZZ(G, H)
COMMON A(10), B, X(2, 4), Q
.....
ZZZ = .....
RETURN
END

Subprograma 4
SUBROUTINE WWW(F, G, H)
B = 0.0
.....
RETURN
END
```

Fig. 11-2

función ZZZ, contienen la misma proposición COMMON y, por tanto, tendrán acceso a un área de memoria común. En particular, cada uno de los programas puede usar o alterar los valores almacenados allí. Por otra parte, la subrutina WWW no tiene una proposición COMMON y, en consecuencia, no tendrá acceso al área de almacenamiento común.

Hay otro punto que deseamos poner en evidencia. La proposición

```
B = 0.0
```

en la subrutina WWW no tiene efecto sobre el área de almacenamiento común por cuanto B es local para el subprograma WWW. Sin embargo, si esta variable apareciera en cualquiera de los otros subprogramas, entonces, cuando se ejecute la proposición B=0.0, se alteraría el valor de B en el área de almacenamiento común. Más aún, este valor de B podría ser utilizado posteriormente por cualquiera de los otros subprogramas.

Nota A. El uso de proposiciones COMMON no rotuladas (blanco) crea solamente *una* área de almacenamiento común que es accesible por diferentes subprogramas. Sin embargo, las localizaciones de memoria de esta área común pueden tener diferentes nombres en los diferentes subprogramas. Por ejemplo, pensemos que el subprograma XXX anterior tiene la siguiente forma:

```
SUBROUTINE XXX(F, K)
COMMON AA(4, 2), BB, CC(11)
.....
RETURN
END
```

Aunque la proposición COMMON se refiere a la misma área de almacenamiento común, las localizaciones de memoria de esta área común se llaman con diferentes nombres en los diferentes programas, como se indica en la figura 11-3. Luego si la proposición

```
CC(2) = 5.5
```

aparece en la subrutina XXX y se ejecuta en la subrutina, el valor de B, en los subprogramas YYY y ZZZ y en el programa principal, se cambia a 5.5.

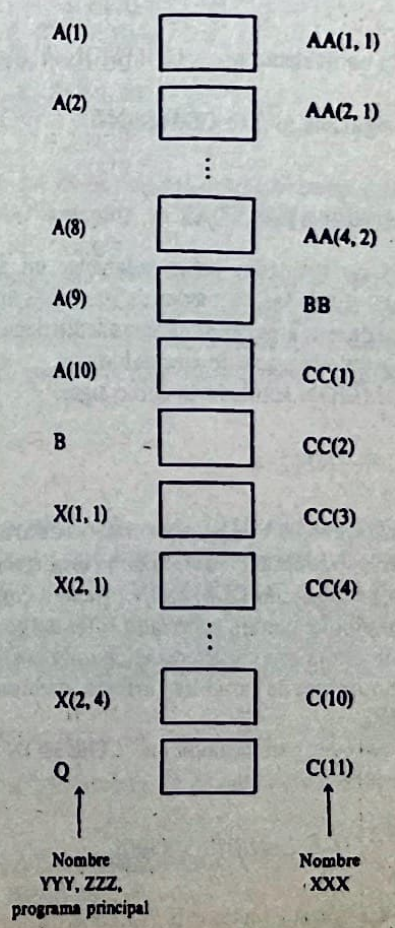


Fig. 11-3

Nota B. El número de celdas de memoria definidas en el área COMMON en un subprograma puede ser más pequeño que el área COMMON, definida en el programa principal. Por ejemplo, se puede permitir lo siguiente:

```
Programa principal:   COMMON X, A(5), Y
     Subprograma:     COMMON B(2, 2)
```

Más aún, las variables compartirán las celdas de memoria en el área COMMON, como sigue:

Nombre en el programa principal:	X	A(1)	A(2)	A(3)	A(4)	A(5)	Y
Nombre en el subprograma:	B(1, 1)	B(2, 1)	B(1, 2)	B(2, 2)			

Obsérvese que los primeros elementos en las proposiciones COMMON X y B(1, 1), comparten la misma localización. Por otra parte, el número de celdas de memoria en el área COMMON de un subprograma nunca puede exceder las del programa principal. Por ejemplo, no se permite,

```
Programa principal:   COMMON C(2, 3)
     Subprograma:     COMMON D(8)
```

Nota C. Se pueden usar varias proposiciones COMMON no rotuladas en un programa. Sin embargo, esto es equivalente a una sola proposición COMMON más grande. Por ejemplo,

```
COMMON A(10), T
COMMON C(2)        es equivalente a    COMMON A(10), T, C(2)
```

y se reservan 13 localizaciones de memoria al área COMMON.

11.9 PROPOSICIONES COMMON ROTULADAS

Las proposiciones COMMON no rotuladas crean solamente un área de almacenamiento común. Utilizando proposiciones COMMON rotuladas, se pueden definir varias áreas de almacenamiento comunes. Como su nombre lo indica, cada una de esas áreas de almacenamiento tiene un rótulo (nombre). Estas proposiciones, como las COMMON no rotuladas, son no ejecutables.

Una proposición típica de COMMON rotulado es como sigue:

```
COMMON /A/ TAX(10), X, INF(2, 4)
```

La proposición no solamente declara que TAX e INF son nombres de arreglos, sino que también define un área de almacenamiento común rotulado (denominado) A con 19 localizaciones de memoria, como se ilustra en la figura 11-4. Obsérvese que la proposición COMMON rotulada empieza con la palabra COMMON seguida por un rótulo (nombre) del bloque común encerrado entre diagonales. (El rótulo, como cualquier otro nombre de variable, consiste de uno a seis caracteres alfanuméricos, el primero de los cuales debe ser alfabético.) Después viene la lista de nombres de variables y arreglos dados a las localizaciones de memoria en esta área de almacenamiento común.

Se pueden definir varias áreas de almacenamiento COMMON rotuladas en una proposición COMMON. Por ejemplo,

```
COMMON /A/ TAX(10), X, INF(2, 4)/B/POP(100)
```

define dos áreas comunes, una rotulada con A y otra con B. Notemos que los arreglos deben declararse en una proposición DIMENSION o en una proposición COMMON, pero no en ambas. Luego cualquiera de las siguientes formas es equivalente a la proposición anterior.

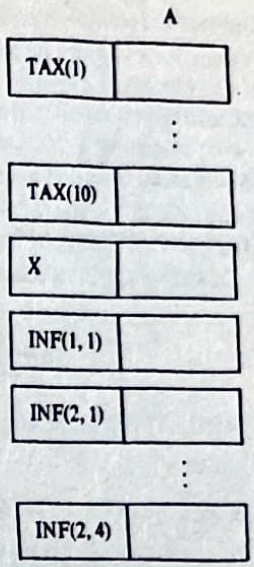


Fig. 11-4

```
DIMENSION TAX(10), INF(2, 4), POP(100)
COMMON /A/TAX, X, INF /B/POP
```

o

```
DIMENSION TAX(10)
COMMON /A/TAX, X, INF(2, 4)
COMMON/B/POP(100)
```

Hagamos hincapié en el hecho de que no hay comas que separen las diferentes áreas comunes rotuladas en una proposición COMMON (por ejemplo, no hay coma precediendo a /B/ en cualquiera de las proposiciones anteriores.)

La ventaja principal de las áreas comunes rotuladas es que diferentes subprogramas pueden tener acceso a diferentes áreas comunes. Es muy importante recordar que el *rótulo* debe ser comunicado de un subprograma a otro. Los nombres de las celdas de memoria dentro de un bloque común particular son locales a ese subprograma y, por lo tanto, pueden ser diferentes, como se estudió en el caso no rotulado. Ilustremos esto con un ejemplo.

EJEMPLO 11.2

Considérese el esqueleto de programa de la figura 11-5. Observe que están definidas dos áreas comunes y sus rótulos son AX y AY. La subrutina SUB1 tiene acceso al área común AY que contiene ocho localizaciones de memoria, y los nombres de estas localizaciones en la subrutina son los mismos nombres del programa principal, esto es, X(1, 1), X(2, 1), . . . , X(2,4). La subrutina SUB 2 comparte con el programa principal, el área común rotulada AX, con 11 localizaciones de memoria. Esas localizaciones se llaman en el programa principal A(1), . . . , A(10), B, pero se llaman T(1, 1), . . . , T(2, 3), S(1), . . . , S(5) en la subrutina, como se muestra en la figura 11-6.

Concluyamos con otros detalles acerca de las proposiciones COMMON rotuladas.

1. Si en un subprograma los nombres de dos áreas comunes son idénticos, tal como en

```
COMMON /A/T(2, 4)/B/X, Y, Z /A/Q(10)
COMMON /B/W(10)
```

entonces el efecto es acumulativo, es decir, la proposición anterior es la misma que

```
Programa principal
DIMENSION X(2, 4)
COMMON/AX/A(10),B /AY/X
.....
CALL SUB1
CALL SUB2(C, D)
.....
END

Subprograma 1
SUBROUTINE SUB1
COMMON /AY/X(2, 4)
.....
RETURN
END

Subprograma 2
SUBROUTINE SUB2(E, F)
COMMON/AX/T(2, 3), S(5)
.....
RETURN
END
```

Fig. 11-5

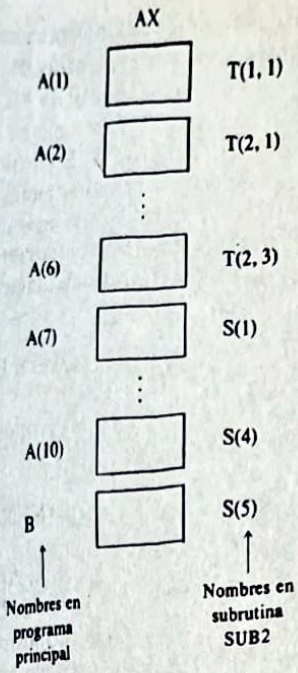


Fig. 11-6

```
COMMON /A/T(2, 4), Q(10) /B/X, Y, Z, W(10)
```

2. El número total de localizaciones de memoria en un área común rotulada, debe ser el mismo en todos los subprogramas que tienen acceso a ella. Observe que esto es diferente en el caso del área no rotulada.
3. Las áreas comunes rotuladas y no rotuladas pueden definirse en el mismo programa, utilizando la misma proposición COMMON. Por ejemplo, la proposición

```
COMMON TAX(100), A, B /Q/X(25, 10)
```

crea un área común no rotulada con 102 localizaciones y un área común rotulada llamada Q con 250 localizaciones. Equivalentemente, podríamos usar

```
COMMON /Q/X(25, 10)//TAX(100), A, B
```

Obsérvese que ningún rótulo aparece entre las diagonales para la segunda área de almacenamiento; luego crea un área común no rotulada, como se vio anteriormente. Esta es la razón por la que una proposición COMMON no rotulada se llama también proposición COMMON blanca.

El uso más general de proposiciones COMMON rotulados y no rotulados puede ser extremadamente complicado y confuso; por consiguiente, consideramos que esa discusión se escapa del marco general de este texto.

11.10 PROPOSICIONES EQUIVALENCE

Supongamos que después de escribir y perforar un programa muy extenso, un programador nota que se han usado diferentes nombres, digamos, MAX, LARGE Y JBIG, para la misma cantidad. En lugar de cambiar todos los nombres y volver a perforar las tarjetas, el programador puede, simplemente, adicionar la proposición EQUIVALENCE

```
EQUIVALENCE (MAX, LARGE, JBIG)
```

al comienzo del programa. Esta proposición informa al computador que los hombres MAX, LARGE y JBIG, se refieren a la misma localización de memoria.

Obsérvese que la proposición EQUIVALENCE comienza con la palabra EQUIVALENCE seguida por paréntesis que encierran las variables que se refieren a la misma localización de memoria.

La proposición EQUIVALENCE también puede usarse para ahorrar memoria, al asignar diferentes nombres a la misma localización de memoria, teniendo cuidado de que estas variables se usen en diferentes partes del programa. Por ejemplo, supongamos que se calcula e imprime el promedio AVE, la media XMED, y el mayor BIG de 100 números. Si las variables AVE, XMED y BIG no se necesitan nunca al mismo tiempo, entonces se puede conservar espacio de almacenamiento asignando las tres variables a la misma localización mediante

```
EQUIVALENCE(AVE, XMED, BIG)
```

Como es usual, se puede usar una proposición EQUIVALENCE para denotar diferentes equivalencias. Por ejemplo,

```
EQUIVALENCE(A, X, Z), (NEXT, LAST), (C, D, E, F)
```

le dice al compilador que A, X y Z compartirán la misma localización de memoria, que NEXT y LAST comparten la misma localización y que C, D, E y F comparten la misma localización de memoria.

La principal aplicación de la proposición EQUIVALENCE ocurre con los arreglos, puesto que se pueden ahorrar miles de posiciones de memoria. Sin embargo, la forma de compartir espacios de almacenamiento en los arreglos puede ser engañosa; por lo tanto, las proposiciones EQUIVALENCE que involucran arreglos, no son tan simples como para las variables no subindicadas estudiadas anteriormente.

Por ejemplo, supongamos que A, B y C son arreglos de un programa declarados por

```
DIMENSION A(50, 100), B(25, 50), C(3750)
```

Supongamos, además, que el arreglo A se requiere solamente en la parte inicial del programa y que no se usa en los cálculos de los arreglos B y C. Podemos ahorrar localizaciones de memoria, almacenando los arreglos B y C en las localizaciones de memoria del arreglo A. En consecuencia, indicaremos que el primer elemento de A y el primer elemento de B están en la misma localización de memoria

```
EQUIVALENCE(A(1, 1), B(1, 1))
```

Puesto que los elementos del arreglo son contiguos, esta proposición también informa al computador que las primeras 1250 localizaciones de B. (¿Puede usted dar la correspondencia?) Podemos usar las 3750 localizaciones restantes del arreglo A para el arreglo C, mediante

```
EQUIVALENCE(A(1, 26), C(1))
```

Esas dos proposiciones se pueden combinar en una sola

```
EQUIVALENCE(A(1, 1), B(1, 1)), (A(1, 26), C(1))
```

Las áreas de almacenamiento están ahora superpuestas y solamente serán asignadas 5000 localizaciones, en lugar de 10,000.

Detalles cruciales para recordar acerca de los arreglos son:

1. Los elementos de un arreglo dado están siempre seguidos uno del otro en un orden lineal dado y que esto no puede cambiarse por una proposición EQUIVALENCE.
2. Solamente pueden aparecer elementos del arreglo en una proposición EQUIVALENCE, pero no el nombre del arreglo en sí.

Nota. Recalcamos que los arreglos están almacenados linealmente en la memoria aunque los arreglos pueden ser multidimensionales. Algunos compiladores permiten utilizar la posición lineal de los elementos de un arreglo en una proposición EQUIVALENCE. Por ejemplo, A(1, 1) ocupa la primera posición en el arreglo

anterior y A(1, 26) ocupa la posición 1251 en el arreglo A. También, B(1, 1) ocupa la primera posición en el arreglo B. En consecuencia,

```
EQUIVALENCE(A(1, 1), B(1, 1)), (A(1, 26), C(1))
```

puede reemplazarse por

```
EQUIVALENCE(A(1), B(1)), (A(1251), C(1))
```

La ventaja de utilizar posiciones lineales es la facilidad de calcular la posición.

En ambas formas anteriores, es importante anotar que los subíndices deben ser constantes, por cuanto la proposición EQUIVALENCE es no ejecutable y las localizaciones de memoria se asignan durante la compilación.

Seguidamente damos unos ejemplos que involucran arreglos con proposiciones EQUIVALENCE.

EJEMPLO 11.3

(*a*) Considerar las proposiciones

```
DIMENSION A(2, 3)
EQUIVALENCE(A(2), X, Y), (A(5), C)
```

Las variables comparten almacenamiento como sigue:

A(1, 1)	A(2, 1) X Y	A(1, 2)	A(2, 2)	A(1, 3) C	A(2, 3)

Esto es, A(2, 1), X, Y son asignadas a la misma localización de almacenamiento y A(1, 3) y C son asignadas a la misma localización de almacenamiento.

(*b*) Supongamos que se puede usar la posición lineal de un arreglo. Consideremos las proposiciones

```
DIMENSION A(2, 3), B(2, 2), C(5)
EQUIVALENCE(A(1), B(1), C(1))
```

Los arreglos comparten espacios de almacenamiento, así:

A(1, 1) B(1, 1) C(1)	A(2, 1) B(2, 1) C(2)	A(1, 2) B(1, 2) C(3)	A(2, 2) B(2, 2) C(4)	A(1, 3) C(5)	A(2, 3)

Obsérvese que C(4) es forzado a compartir una localización con A(2,2) y B(2, 2) puesto qué originalmente asignamos los primeros elementos de cada arreglo a la misma localización.

(*c*) Considerar las proposiciones

```
DIMENSION A(2, 3), C(7)
EQUIVALENCE(A(3), C(1))
```

Entonces los arreglos aparecen en la memoria, como sigue:

A(1, 1)	A(2, 1)	A(1, 2) C(1)	A(2, 2) C(2)	A(1, 3) C(3)	A(2, 3) C(4)	C(5)	C(6)	C(7)

Obsérvese que el arreglo se extiende hacia afuera por el lado derecho.

Se puede también usar una proposición EQUIVALENCE en conjunto con una proposición COMMON; la proposición EQUIVALENCE puede solamente alargar el almacenamiento común, pero no puede cambiar las localizaciones originales de las variables. Ilustremos con ejemplos:

EJEMPLO 11.4

(*a*) Considerar las proposiciones

```
DIMENSION A(4), C(2), D(8)
COMMON A, B, C
EQUIVALENCE(A(3), D(1))
```

El área de almacenamiento común aparecerá como sigue:

```
A(1)  A(2)  A(3)  A(4)   B    C(1)  C(2)
            D(1)  D(2)  D(3)  D(4)  D(5)  D(6)  D(7)  D(8)
```

(Por conveniencia notacional no hemos dibujado las cajas con los nombres de las variables, como lo hicimos en el ejemplo 11.3.) La proposición original COMMON agrupa siete localizaciones en almacenamiento común. La proposición EQUIVALENCE extiende el almacenamiento común a 10 localizaciones.

(*b*) No son permitidas las siguientes proposiciones FORTRAN.

```
DIMENSION A(4), C(2), D(8)
COMMON A, B, C
EQUIVALENCE(A(1), D(3))
```

Según el razonamiento anterior, compartir el área común, se vería como sigue:

```
                A(1)  A(2)  A(3)  A(4)   B    C(1)  C(2)
D(1)  D(2)  D(3)  D(4)  D(5)  D(6)  D(7)  D(8)
```

Aquí, A(1) ocupa la tercera localización aunque por la proposición COMMON debería ocupar la primera localización. Puesto que FORTRAN no nos permite cambiar las posiciones originales de las variables en el almacenamiento común, esas proposiciones tampoco pueden permitirse.

Otros usos de la proposición EQUIVALENCE, especialmente si involucran proposiciones COMMON, pueden ser muy complicados y dependen de la máquina; por lo tanto, consideramos que este examen escapa de los límites de este texto. Además, puesto que es completamente posible trabajar sin una proposición EQUIVALENCE, advertimos a los programadores principiantes para que eviten el uso de la proposición EQUIVALENCE siempre que sea posible.

11.11 BLOCK DATA

Algunos compiladores no permiten utilizar la proposición DATA para inicializar valores de variables en áreas COMMON rotuladas, mientras que otros permiten inicialización de cualquier área COMMON. Sin embargo, un subprograma llamado BLOCK DATA se puede utilizar para este propósito.

Un subprograma BLOCK DATA únicamente puede consistir de las siguientes proposiciones no ejecutables:

```
DATA
COMMON
DIMENSION
EQUIVALENCE
```

y cualquier declaración de tipo. Ilustremos cómo trabaja este programa, con un ejemplo.

```
DIMENSION X(20), Y(15)
COMMON /A/X, U, V /B/Y
          :
END
```

Supóngase que deseamos que cada elemento de X e Y sea inicializado con 0.0. No se permite el uso de una proposición DATA, puesto que X e Y están en áreas COMMON rotuladas. Obviamente, se pueden usar proposiciones de asignación para la inicialización lo que requerirá tiempo de ejecución. Por otra parte, se puede usar el siguiente subprograma BLOCK DATA:

```
BLOCK DATA
DIMENSION X(20), Y(15)
COMMON /A/X, U, V /B/Y
DATA X/20*0.0/, Y/15*0.0/
END
```

Resumamos el uso del subprograma BLOCK DATA:

1. El subprograma siempre empieza con la proposición BLOCK DATA.
2. Deben estar listadas en el subprograma todas las variables en el área COMMON, aunque se desee inicializar solamente algunas de las variables.
3. Se puede usar solamente un subprograma BLOCK DATA para inicializar variables en cualquier área COMMON.

11.12 EXTERNAL

Recalquemos que la comunicación entre un llamado de programa y un subprograma (además de COMMON y EQUIVALENCE) se da sólo mediante la lista de argumentos. Algunas veces puede ser deseable utilizar un nombre de un subprograma como un argumento real en la proposición de llamado para otro subprograma. El siguiente esqueleto de programa ilustra esta situación:

```
        Programa principal
        ..........
        EXTERNAL TOTAL
        X = BAL(ASSET, PAY, TOTAL)
        ..........
Subprograma } END
        FUNCTION BAL(A, B, C)
        T = C(A)
        .........
        RETURN
Subprograma } END
        FUNCTION TOTAL(Z)
        ..........
        RETURN
        END
```

Como indica una de las flechas, el nombre del subprograma TOTAL es transferido al subprograma BAL utilizando el parámetro C. En consecuencia, C(A) se calcula en BAL usando el subprograma TOTAL, como se indica por otra flecha.

Sin embargo, sin una declaración especial, el compilador no podría distinguir entre el nombre de un subprograma y una simple variable que transporta un valor. Para evitar la confusión, se usa una declaración EXTERNAL en el llamado del programa para indicar que el nombre del subprograma usado en el argumento es externo a ese subprograma.

Problemas

PROPOSICIONES DE DECLARACION DE TIPO, PROPOSICION IMPLICIT

11.1 Encuentre los errores, si los hay, en cada uno de los segmentos:

(*a*) INTEGER, RATE, TIME, DISTANCE
(*b*) DIMENSION R(4), S(5, 3)
DOUBLE PRECISION A(7), B, R(8), S,
(*c*) IMPLICIT, COMPLEX (A – D, F, G) LOGICAL(C, X – Z)
(*d*) IMPLICIT REAL M, N, COMPLEX P – R
(*e*) DOUBLE PRECISION NUMBER(4, 7) X, Y, TEST(8, 15) L, M

11.2 Considere el siguiente segmento:

```
IMPLICIT COMPLEX(H – K) LOGICAL(P, R)
REAL COEF, KAPPA, LAMBDA
INTEGER RATE, TIME
```

e indique el tipo de cada variable

(*a*)	ALPHA	(*d*)	KOUNT	(*g*)	LAST	(*j*)	ROOM
(*b*)	INDEX	(*e*)	PARTY	(*h*)	LAMBDA	(*k*)	TIME
(*c*)	KAPPA	(*f*)	QUEUE	(*i*)	RATE	(*l*)	TARGET

DOBLE PRECISION

11.3 Indique si cada uno de los siguientes valores representa una constante real (precisión sencilla) o una constante de doble precisión (aceptando siete dígitos significativos para las constantes reales):

(*a*)	1.45D + 03	(*d*)	12345678	(*g*)	123.123123 + 0.E0
(*b*)	5.9321E – 7	(*e*)	23.4 + 0.D0	(*h*)	5.55E – 26 + 58.4
(*c*)	3456.765432	(*f*)	111.222333E – 5	(*i*)	6.7 + 4.5D1 – 7.8E3

11.4 Determine la salida de cada uno de los segmentos de programa:

(*a*)
```
      DOUBLE PRECISION DA
      A = 0.555666777888999
      DA = 0.555666777888999
      WRITE(6, 10) A, A
   10 FORMAT(1X, E25.3, 5X, E25.9)
      WRITE(6, 20) DA, DA
   20 FORMAT(1X, D25.3, 5X, E25.9)
      STOP
      END
```

(*b*)
```
      DOUBLE PRECISION X, Y, Z
      X = 70/9
      Y = 70.0/9.0
      Z = 70.0D0/9.0D0
      WRITE(6, 30) X, Y, Z
   30 FORMAT(1X, D20.12)
```

11.5 Suponga que A=9999999.0, B=125.0 y C=9999998.0. Tenga en cuenta que

X = A + B – C = 126.0

(lo cual puede obtenerse utilizando aritmética de doble precisión).

(*a*) Calcule X, utilizando aritmética de precisión sencilla (conviniendo siete cifras significativas).
(*b*) Encuentre el error y porcentaje de error.

11.6 Escriba un programa que calcule *n*! para $n = 1, 2, \ldots, 15$. (Debe utilizar aritmética de doble precisión, ya que ni la aritmética entera ni la aritmética de precisión sencilla tienen exactitud suficiente.)

11.7 Recordamos que el área de un triángulo, cuyos lados tienen longitudes a, b y c está dada por

$$\text{AREA} = \sqrt{s(s-a)(s-b)(s-c)}$$

donde

$$s = (a+b+c)/2$$

Escriba un programa que lea los valores de a, b y c y calcule e imprima el AREA, utilizando aritmética de doble precisión.

NUMEROS COMPLEJOS

11.8 Considere $z=2-3i$ y $w=4+5i$. Encuentre algebraicamente

(*a*) $z+w$ (*b*) zw (*c*) z/w (*d*) $\bar{z}$ y $\bar{w}$ (*e*) $|z|$ y $|w|$

Verifique los resultados por computador, utilizando un programa Fortran.

11.9 Suponiendo que X, Y y Z son variables complejas y que A, B y C son variables reales, encuentre los errores, si los hay, en cada una de las proposiciones

(*a*) X = (3, 4.5) (*c*) Z = 3.6 + (5.0, 7.6)
(*b*) Y = (A, B + C) (*d*) A = COMPLEX(B, C)

11.10 Suponga que se perforan dos tarjetas de datos como sigue:

```
                1         2
       123456789012345678901234 5
Primera tarjeta  1.1  2.2  3.3  4.4

Segunda tarjeta  5.5  6.6  7.7  8.8
```

Encuentre la salida de cada uno de los segmentos de programa:

(*a*)
```
      COMPLEX A, B, C
      READ(5, 10) A, B
   10 FORMAT(2F5.1)
      C = A*B
      WRITE(6, 20) C
   20 FORMAT(1X, F10.2)
```

(*b*)
```
      COMPLEX A, B, C
      READ(5, 50) A, B
   50 FORMAT(3F5.1)
      C = A*B
      WRITE(6, 60) C
   60 FORMAT(1X, 'LA PARTE REAL ES', 7X, F10.2/1X,
     1'LA PARTE IMAGINARIA ES', 2X, F10.2)
```

11.11 Escriba un programa que lea tres números complejos e imprima los tres números en una línea y su suma y producto en una segunda línea.

11.12 Escriba un programa que lea los números reales a, b y c, y resuelva la ecuación cuadrática

$$ax^2 + bx + c = 0$$

Utilice la fórmula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

(Obsérvese que las raíces son complejas si el discriminante $D=b^2-4ac$ es menor que cero.)

PROPOSICIONES GO TO ASIGNADA, ENTRADAS Y RETORNOS MULTIPLES

11.13 Encuentre los errores, si los hay, en cada una de las proposiciones Fortran:

(*a*) ASSIGN +45 TO NUMBER
(*b*) ASSIGN N TO NUMBER
(*c*) GO TO NUMBER (5, 78, 6, −43)
(*d*) GO TO 200, (75, 25, 200, 50)

11.14 Encuentre los errores, si los hay, en cada uno de los segmentos de programa Fortran:

(*a*)
```
ASSIGN 25 TO NEXT
GO TO NEXT, (50, 75, 300)
```

(*b*)
```
ASSIGN 25 TO NEXT
NEXT = 50
GO TO NEXT, (25, 40, 50, 70)
```

11.15 Encuentre los errores, si los hay, en el siguiente esqueleto de programa:

Programa principal
```
.....
CALL BIG(X, Y, Z)
.....
CALL MIDDLE(P, A + B)
.....
CALL RATE(X + Y, J, N + 4)
.....
STOP
END
```

Subprograma
```
SUBROUTINE BIG(A, B, C)
.....
ENTRY MIDDLE(X, Y, Z)
.....
ENTRY RATE(C, D, K)
.....
END
```

11.16 Considere la siguiente proposición CALL y la correspondiente proposición SUBROUTINE:

```
CALL UPDATE(A, 88, B, 36, C, N, 44)
SUBROUTINE UPDATE(X, *, Y, *, Z, NUMBER, *)
```

Determine el punto al cual se transfiere el control si se encuentra la siguiente proposición en el subprograma UPDATE:

(*a*) RETURN 1 (*b*) RETURN 2 (*c*) RETURN 3 (*d*) RETURN

COMMON, EQUIVALENCE

11.17 Encuentre los errores, si los hay, en cada una de las proposiciones Fortran:

(*a*) COMMON (X, Y, Z)
(*b*) EQUIVALENCE X, Y, Z
(*c*) COMMON X, Y(2, 4), Z(8), W,
(*d*) EQUIVALENCE (X, Y) (A(1), B(4), Z)
(*e*) COMMON /A/ NUM(5), X, /B/ INDEX(4, 5)

11.18 Dibuje el esquema de almacenamiento que resulta del siguiente esqueleto de programa:

Programa principal
```
COMMON Y(4), A, B, C(2)
.....
CALL NNN(P, Q, R)
.....
END
```

Subprograma
```
SUBROUTINE NNN(P1, P2, P3)
COMMON D, X(2), E(2), F(2)
.....
END
```

11.19 Determine el esquema de almacenamiento que resulta de los siguientes programas:

(*a*)
```
DIMENSION A(5), B(4), C(7)
EQUIVALENCE (A(1), B(1), C(2))
```

(*b*)
```
DIMENSION X(2, 3), Y(3, 2)
EQUIVALENCE(X(3), Y(1))
```

11.20 Suponga que A es una matriz de 3 × 3. Queremos que DIAG sea un arreglo lineal de 3 elementos que denote las entradas diagonales a A. Discuta el error en el siguiente segmento de programa:

```
DIMENSION A(3, 3), DIAG(3)
EQUIVALENCE (A(1), DIAG(1)), (A(5), DIAG(2)), (A(9), DIAG(3))
```

Respuestas a problemas seleccionados

11.1 (*a*) No debe haber coma después de INTEGER.
(*b*) R se declara dos veces con diferentes tamaños. No debe haber coma después de S.
(*c*) No debe haber coma después de IMPLICIT. Debe haber coma antes de LOGICAL.
(*d*) IMPLICIT REAL(M, N), COMPLEX(P – R)
(*e*) Debe haber coma antes de X y L.

11.2 (*a*) real (*b*) complejo (*c*) real (*d*) complejo (*e*) lógica (*f*) real
(*g*) entero (*h*) real (*i*) entero (*j*) lógica (*k*) entero (*l*) real

11.3 (*a*) Doble precisión (*b*) real (*c*) doble precisión (*d*) Ninguna, puesto que es entera.
(*e*) doble precisión (*f*) doble precisión (*g*) doble precisión (*h*) real (*i*) doble precisión

11.4 (*a*) 0.556E 00 0.555666700E 00
0.556D 00 0.555666778D 00

(*b*) 0.700000000000D 01
0.777777700000D 01
0.777777777778D 01

11.5 (*a*) 122.0 (*b*) Error=4, el cual es 3%.

11.7 Todas las variables que se van a utilizar deben declararse como de doble precisión; debe utilizarse la función raíz cuadrada de doble precisión DSQRT:

```
      DOUBLE PRECISION A, B, C, S, AREA
      READ(5, 10) A, B, C
10    FORMAT(3D20.10)
      S = (A + B + C)/2.0
      AREA = DSQRT(S*(S – A)*(S – B)*(S – C))
      WRITE(6, 20) AREA
20    FORMAT(1X, 'EL AREA ES', 3X, D20.12)
      STOP
      END
```

11.8 (*a*) $6 + 2i$ (*b*) $23 - 2i$ (*c*) $-7/41 - 22i/41$ (*d*) $2 + 3i, 4 - 5i$ (*e*) $\sqrt{13}, \sqrt{41}$

11.9 (*a*) El 3 debe escribirse como 3.0.
(*b*) CMPLX(A, B+C), y no (A, B+C).
(*c*) Correcta.
(*d*) CMPLX(B, C) y no COMPLEX(B, C).

11.10 (*a*) −8.47 (*b*) LA PARTE REAL ES −8.47
19.36 LA PARTE IMAGINARIA ES 13.31

11.13 (*a*) 43 debe estar sin signo, ya que es un número de proposición.
(*b*) N debe ser un número de proposición y no una variable entera.
(*c*) Debe haber una coma después de NUMBER. Además, —43 no está permitido.
(*d*) El primer 200 debe ser una variable entera y no una constante entera.

11.14 (*a*) El valor 25 de la variable NEXT debe aparecer en la lista con que aparece NEXT en la proposición GO TO.
(*b*) El valor de NEXT puede cambiarse solamente por medio de otra proposición ASSIGN, por ejemplo,

```
ASSIGN 30 TO NEXT
```

No puede cambiarse por una proposición de asignación.

11.15 En el programa principal MIDDLE tiene solamente dos argumentos, mientras que en el subprograma tiene tres. El segundo argumento J en RATE debe ser real, ya que debe corresponder con el parámetro real D.

11.16 (*a*) A la proposición rotulada como 88 en el llamado de programa.
(*b*) A la proposición numerada como 36 en el llamado de programa.
(*c*) A la proposición rotulada como 44 en el llamado de programa.
(*d*) A la primera proposición ejecutable que sigue a la proposición CALL.

11.17 (*a*) Las variables no deben estar encerradas entre paréntesis.
(*b*) Las variables deben estar encerradas entre paréntesis.
(*c*) No debe haber coma después de W.
(*d*) Debe haber una coma después de (X, Y).
(*e*) No debe haber coma después de X.

11.18

Nombre en el programa principal:	Y(1)	Y(2)	Y(3)	Y(4)	A	B	C(1)	C(2)
Nombre en el subprograma:	D	X(1)	X(2)	E(1)	E(2)	F(1)	F(2)	

11.19 (*a*)

A(1)	A(2)	A(3)	A(4)	A(5)			
B(1)	B(2)	B(3)	B(4)				
C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	

(*b*)

X(1, 1)	X(2, 1)	X(1, 2)	X(2, 2)	X(1, 3)	X(2, 3)			
		Y(1, 1)	Y(2, 1)	Y(3, 1)	Y(1, 2)	Y(2, 2)	Y(3, 2)	

11.20 Las tres equivalencias indican que DIAG(1) debe compartir una localización con A(1, 1), DIAG(2) con A(2, 2) y DIAG(3) con A(3, 3) como sigue:

A(1, 1)	A(2, 1)	A(3, 1)	A(1, 2)	A(2, 2)	A(3, 2)	A(1, 3)	A(2, 3)	A(3, 3)
DIAG(1)				DIAG(2)				DIAG(3)

Sin embargo, lo anterior no cumple la condición de que los elementos de un arreglo deben aparecer contiguos en cada localización, esto es, no podemos separar los elementos de DIAG.

Capítulo 12

Fortran estructurado

Aún con muchos años de práctica y experiencia, es muy difícil dominar el arte de la programación. Los principiantes se ven, algunas veces, muy confundidos y desanimados porque sus programas se tornan a menudo en meras colecciones de proposiciones sin lógica y sin organización aparentes.

La principal fuente de confusión y error en programación Fortran puede deberse a: (1) el confuso control de estructuras en el mismo lenguaje fortran, y (2) al uso indisciplinado de algunas de las construcciones en Fortran, principalmente las proposiciones GO TO.

A través de los años se han desarrollado ciertos principios para lograr una buena práctica de programación, lo que ha originado la noción de programación estructurada. En concordancia, muchos mecanismos precisos de control estructurado se han incorporado en los preprocesadores Fortran. Estas construcciones Fortran estructuradas ayudan a que el principiante reduzca sus confusiones y errores, y a que los programas escritos con ellas sean más fáciles de leer, comprender y modificar.

Este capítulo discute algunas de las estructuras de control más comunes sus usos y sus relaciones con el Fortran no estructurado. En efecto, los lectores que tengan acceso a procesadores Fortran estructurado pueden estudiar las estructuras IF después del capítulo 4 y las estructuras DO después del capítulo 5.

12.1 ESTRUCTURAS IF

Las estructuras IF en Fortran estructurado tienen algunas variaciones. Sin embargo, cada estructura IF empieza siempre con una *proposición* IF *ablocada* que tiene la forma

IF(*logexp*) THEN

(donde *logexp* es una expresión lógica) y termina con la proposición

ENDIF

Estas dos proposiciones sirven como delimitadores dentro de los cuales va el cuerpo de la estructura. Se discutirán algunas de sus variaciones.

Alternativa simple

Una forma de la estructura IF y su diagrama de flujo correspondiente, se muestra en la figura 12-1.

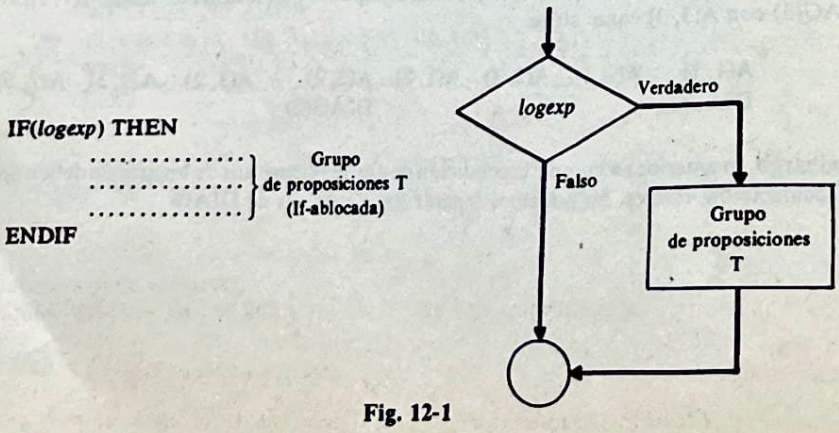


Fig. 12-1

Como lo indica el diagrama de flujo, primero se evalúa la expresión lógica *logexp*. Si es verdadera, se ejecuta el grupo de proposiciones T (llamado el bloque IF); si es falsa, se salta el grupo de proposiciones T y se transfiere el control a la primera proposición, después de la proposición ENDIF.

La estructura IF anterior es equivalente a cualquiera de los segmentos de programas Fortran no estructurado que usan la proposición IF lógica:

```
IF(.NOT.logexp) GO TO 10
          ................ }
          ................ } Grupo de
          ................ } proposiciones
                             T
10 ...............
```

```
IF(logexp) GO TO 20
GO TO 10
20        ........... }
          ........... } Grupo de
          ........... } proposiciones
                         T
10 ...............
```

Observe que el primer segmento emplea el complemento de *logexp* en la proposición IF lógica; esto evita el uso de una proposición adicional GOTO, como en el segundo segmento. El complemento de una expresión lógica se puede obtener fácilmente anteponiendo .NOT. a la expresión. Sin embargo, para los operadores relacionales simples, los complementos se muestran a continuación:

Operador	.EQ.	.NE.	.GT.	.GE.	.LT.	.LE.
Complemento	.NE.	.EQ.	.LE.	.LT.	.GE.	.GT.

Note que el complemento del complemento de una expresión lógica *logexp* es *logexp* mismo; es decir, .NOT.(.NOT. *logexp*) es equivalente a *logexp*.

EJEMPLO 12.1

Los siguientes segmentos Fortran son equivalentes:

```
(a) IF(A.GT.B) THEN        (b)      IF(A.LE.B) GO TO 10
          TEMP = A                        TEMP = A
          A = B                           A = B
          B = TEMP                        B = TEMP
    ENDIF                          10 ....................
```

Una alternativa simple de estructura IF que contiene solamente una proposición simple en su grupo de proposiciones es equivalente a una proposición IF lógica, y viceversa. Por ejemplo, las siguientes son equivalentes:

```
(a) IF(BIG.LT.A) THEN      (b) IF(BIG.LT.A) BIG = A
          BIG = A
    ENDIF
```

Nota. No se permite la transferencia de control de fuera hacia dentro del bloque IF.

Alternativas dobles

Para implementar decisiones que involucren dos alternativas diferentes, se puede insertar la cláusula ELSE en la estructura IF anterior. La forma general de esta estructura y su correspondiente diagrama de flujo se muestran en la figura 12-2.

```
IF(logexp) THEN
          ..........
          ..........  Grupo
          ..........  de proposiciones T
                      (If-ablocada)
ELSE
          ..........
          ..........  Grupo
          ..........  de proposiciones F
                      (Else-ablocada)
ENDIF
```

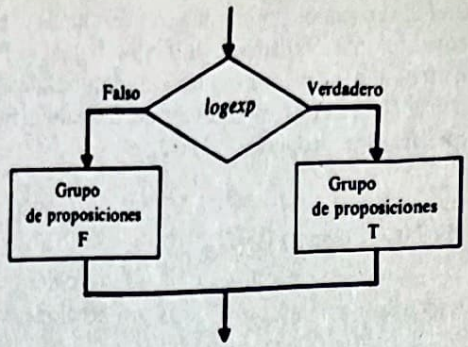


Fig. 12-2

Como indica el diagrama de flujo, se evalúa primero la expresión lógica *logexp*. Si es verdadera, se ejecuta el bloque IF (grupo de proposiciones T); si es falsa, se ejecuta el grupo de proposiciones F (llamado el bloque ELSE). En cualquier caso, después de la ejecución del grupo T o el F, el control se transfiere a la primera proposición ejecutable que se encuentra después de la proposición ENDIF.

La estructura IF anterior es equivalente a cualquiera de los siguientes segmentos Fortran que emplean la proposición IF lógica:

```
(a)     IF(.NOT.logexp) GO TO 10
                ..........
                ..........  Grupo
                ..........  de proposiciones T
        GO TO 99
10              ..........
                ..........  Grupo
                ..........  de proposiciones F
99      ..................
```

```
(b)     IF(logexp) GO TO 20
                ..........
                ..........  Grupo
                ..........  de proposiciones F
        GO TO 99
20              ..........
                ..........  Grupo
                ..........  de proposiciones T
99      ..................
```

Obsérvese que si no se usa el complemento de *logexp*, como en (*b*), se invierte la posición de los grupos de proposiciones; es decir, el grupo F aparece primero y luego el T, mientras que en la estructura IF, el grupo T aparece primero y luego el F.

EJEMPLO 12.2

Los siguientes segmentos Fortran son equivalentes; cada uno calcula el salario bruto de un empleado, a quien se le paga hora y media, por hora extra:

```
(a)     IF(HOUR.LE.40.0) THEN
                GROSS = HOUR*RATE
          ELSE
                GROSS = 40.0*RATE + (HOUR - 40.0)*1.5*RATE
        ENDIF
(b)     IF(HOUR.GT.40.0) GO TO 10
                GROSS = HOUR*RATE
        GO TO 99
10              GROSS = 40.0*RATE + (HOUR - 40.0)*1.5*RATE
99      ..............................
```

Nota. Como antes, no se permite la transferencia de control de fuera hacia dentro del bloque IF, o del ELSE.

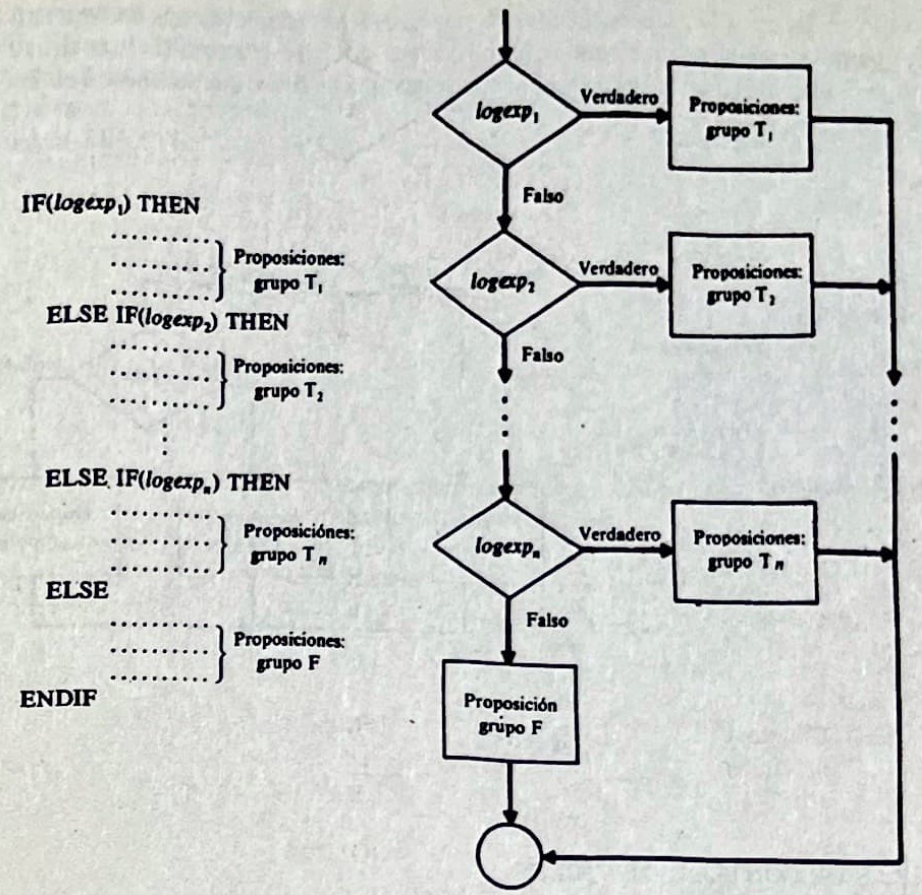


Fig. 12-3

Alternativas múltiples

En la estructura IF básica se pueden insertar varias cláusulas ELSE IF para implementar ciertas decisiones que involucran más de dos alternativas. Tal estructura, conocida como alternativa múltiple, y su correspondiente diagrama de flujo, se muestran en la figura 12-3.

Como lo indica el diagrama de flujo, se evalúan primero, una por una, las expresiones lógicas $logexp_1$, $logexp_2$, ... Si $logexp_K$ es la primera expresión verdadera, se ejecuta el grupo de proposiciones T_k. Si todas las expresiones son falsas, se evalúa el grupo de proposiciones F. En cualquier caso, después de la ejecución de uno de los grupos de proposiciones se transfiere el control a la primera instrucción, después de la proposición ENDIF.

El uso de ELSE es opcional. Es decir, si no existe el grupo de proposiciones F, se omite la palabra clave ELSE.

EJEMPLO 12.3

El siguiente segmento en Fortran estructurado calcula la bonificación de un vendedor. Si el programa fuera escrito en Fortran no estructurado, se requerirían muchas proposiciones GO TO.

```
IF(SALES.LE.200.0) THEN
      BONUS = 0.0
ELSE IF(SALES.LE.1000.0) THEN
      BONUS = SALES*0.05
ELSE
      BONUS = 50.0 + SALES*0.02
ENDIF
```

Finalmente, como en los lazos DO, las estructuras IF no se pueden traslapar. O sea, una estructura IF debe estar completamente contenida en un bloque de proposiciones, de otra estructura IF. Esta situación se ilustra en el programa esquematizado con su correspondiente diagrama de flujo, que se muestra en la figura 12-4.

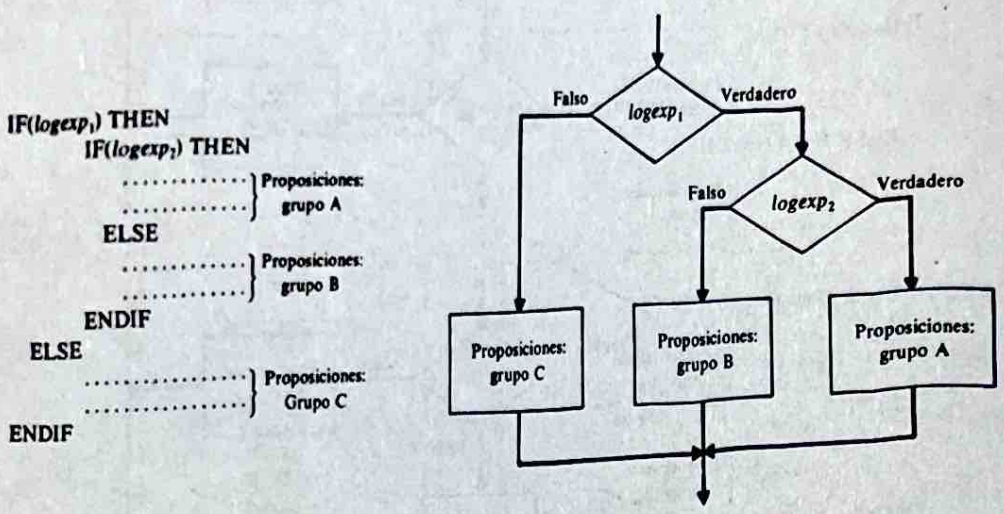


Fig. 12-4

12.2 ESTRUCTURAS DE CONTROL DE CICLOS

Se discutirán tres de estas estructuras: los lazos WHILE, FOR y DO generalizado. Los dos primeros se han implementado en muchos procesadores, incluyendo WATFIV-S, y el tercero se incluye en algunas versiones de Fortran 77.

El lazo DO del Fortran no estructurado, que se discutió en el capítulo 5, ha sido una estructura de control de lazo muy usada y conveniente. Sin embargo, tiene varias restricciones inconvenientes; es decir, la variable de control del lazo (el índice), debe ser de tipo entero, los parámetros del lazo no pueden ser expresiones aritméticas y no se puede decrementar el índice. Estos inconvenientes motivaron la construcción de muchas otras estructuras de lazo. Las estructuras WHILE y FOR, que se discutirán posteriormente, son dos de las más empleadas.

Estructura WHILE

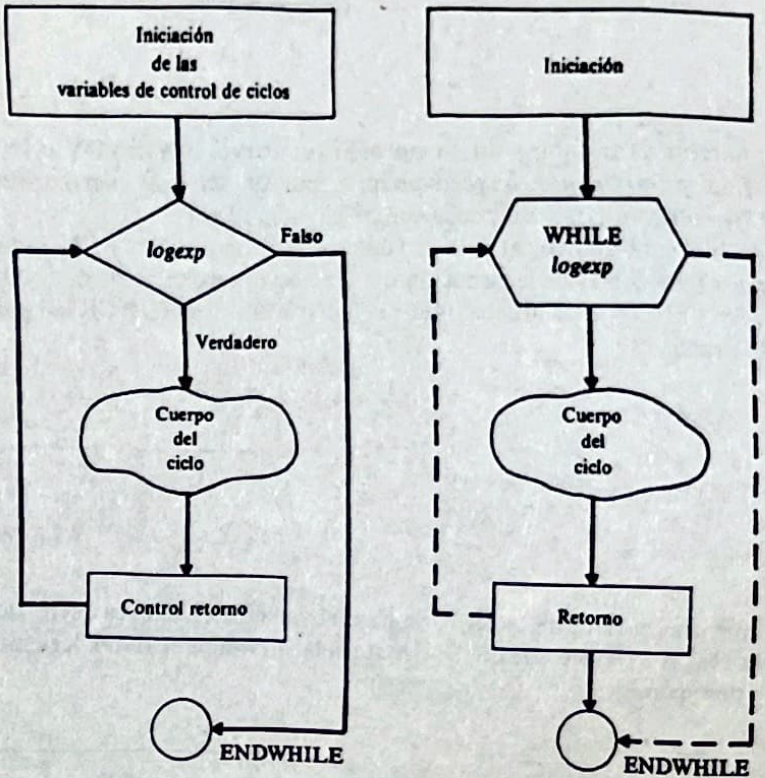
Muchos procesos iterativos no se pueden controlar fácilmente con un contador. Frecuentemente, en tales casos, puede ser conveniente usar la estructura WHILE, que tiene la siguiente forma general:

```
WHILE(logexp) DO
        .....
        .....   Cuerpo del lazo
        .....
ENDWHILE
```

Primero se evalúa la expresión lógica *logexp*. Si es verdadera, se ejecuta el cuerpo del lazo. Además, el cuerpo del lazo se repetirá mientras *logexp* sea verdadero. Cuando *logexp* es falso se salta el cuerpo del lazo y el control se transfiere a la primera proposición después de la proposición ENDWHILE. Observe que las proposiciones WHILE y ENDWHILE sirven como delimitadores, encerrando la estructura iterativa.

Es muy importante recordar que la expresión lógica *logexp* es siempre evaluada antes de ejecutar el lazo.

Normalmente, se pueden involucrar una o más variables en la expresión lógica *logexp* de la estructura WHILE. Estas variables, llamadas variables de control del lazo, se deben inicializar previamente a la ejecución de la estructura WHILE. Para evitar un lazo infinito, los valores de las variables también deben ser actualizadas en cada iteración del lazo. Por lo general, la actualización se hace justamente antes de la proposición REPEAT. Esto se indica en el diagrama de flujo de la figura 12-5, que representa la estructura WHILE.



Implementación de la estructura WHILE

Diagrama de flujo equivalente usando una caja WHILE

Fig. 12-5

EJEMPLO 12.4

Considere el polinomio cuadrático $y = 2x^2 - 3x - 5$. El siguiente programa Fortran, que usa la estructura WHILE, encuentra y para valores de x desde -4 a 4 con pasos de 0.5.

```
      X = -4.0
      WHILE(X.LE.4.0) DO
            Y = 2.0*X**2 - 3.0*X - 5.0
            WRITE(6, 10) X, Y
10          FORMAT(2(3X, F10.3))
            X = X + 0.5
      ENDWHILE
```

Compare la simplicidad de este programa con el programa del problema 5.11(*a*), que hace lo mismo, pero usa el lazo DO básico indizado. Fue necesario hacer esto para poder contar el número de iteraciones y escribir X en términos del índice I.

Estructura FOR

Como se dijo, una restricción severa del lazo DO no estructurado es su incapacidad para decrementar el índice. Otro inconveniente es que el valor final del índice no se prueba hasta después de la primera iteración. Así, aunque M > N, el lazo DO básico (no estructurado) encabezado por la proposición DO

```
DO 50 K = M, N
```

será ejecutado de inmediato.

Estos inconvenientes se eliminan en la estructura FOR, que tiene la siguiente forma:

```
FOR(INDEX = INV, ENDV, INCR) DO
    .....
    .....   Cuerpo del lazo
    .....
ENDFOR
```

INDEX debe ser un nombre de variable a valor entero. Sin embargo, INV, ENDV e INCR que representan *valor inicial*, *valor final* e *incremento*, respectivamente, pueden ser cualquier expresión entera, con la excepción de que el incremento INCR no puede tomar el valor cero.

El diagrama de flujo para implementar la estructura FOR se muestra en la figura 12-6. INDEX es inicializado con el valor INV. Antes de la ejecución del lazo, se prueba el valor de INDEX para ver que no haya pasado el valor final ENDV. Esta prueba difiere según que el valor de INCR sea positivo o negativo. Si INCR es positivo, la prueba es:

Es INDEX ≤ ENDV?

Si INCR es negativo, la prueba será:

Es INDEX ≥ ENDV?

El lazo se ejecutará mientras el valor de INDEX no pase el valor de ENDV, es decir, tantas veces como el resultado de la prueba sea "sí". Después de que se ejecuta el lazo, el valor de INDEX se incrementa en INCR, antes de la siguiente iteración.

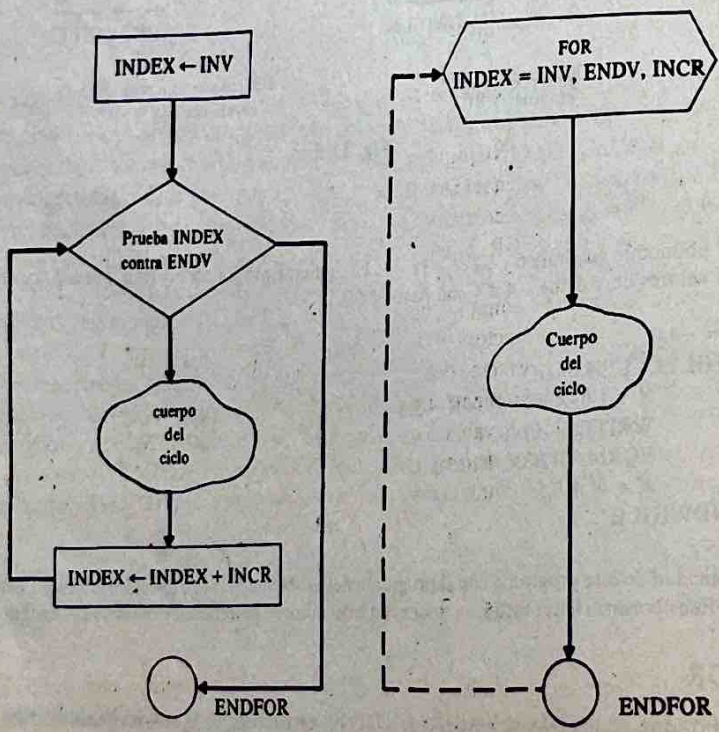


Fig. 12-6

Como el incremento INCR puede ser negativo, INDEX disminuirá en valor. Esta ventaja se ilustra en el siguiente ejemplo.

EJEMPLO 12.5

Suponga que se quieren imprimir los enteros 100, 99, 98,..., 1 en ese orden. Esto se puede hacer de dos maneras: (1) con el lazo DO básico (no estructurado) (ver ejemplo 5.5) y (2) con la estructura FOR:

```
(a)       DO 500 K = 1, 100          (b)       FOR(K = 100, 1, -1) DO
                  J = 101 - K                          WRITE(6, 30) K
                  WRITE(6, 30) J           30          FORMAT(1X, I5)
     30           FORMAT(1X, 5I)                ENDFOR
    500   CONTINUE
```

Obsérvese que en (*b*) el lazo termina cuando K tiene el valor cero, es decir, ha pasado el valor final 1.

Lazo DO generalizado (Lazo DO estructurado)

Las estructuras WHILE y FOR eliminan algunos de los inconvenientes del lazo DO básico (no estructurado) discutido en el capítulo 5. El nuevo Fortran 77 contiene una construcción de lazo DO generalizado, que combina lo mejor de las estructuras WHILE y FOR.

Este lazo DO tiene la misma apariencia básica que el lazo DO indizado, introducido en las secciones 5.3 y 5.4. Esto es, tendrá la siguiente forma:

```
DO n VAR = INV, ENDV, INCR
     .....
     .....   } Cuerpo del lazo DO
     .....
n    CONTINUE
```

De nuevo, el par DO-CONTINUE encierra el cuerpo del lazo por repetir.

VAR, llamada la *variable de control del lazo*, puede ser el nombre de una variable entera o real; los *parámetros del lazo* INV, ENDV e INCR que representan el *valor inicial*, el *valor final* (o valor de prueba) y el *incremento*, respectivamente, pueden ser cualquier expresión entera o real. (Esto contrasta con el lazo DO del capítulo 5, donde el índice debía ser una variable entera y los parámetros constantes enteras positivas o variables enteras.) Si se omite INCR, el incremento se supone 1, porque INCR, no puede ser cero.

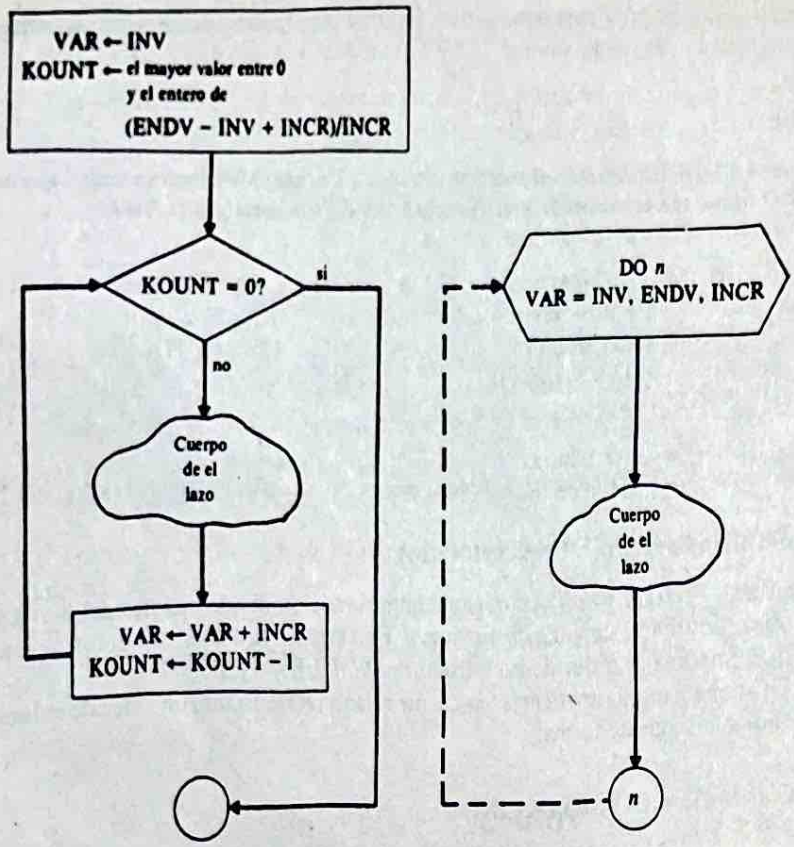
De la discusión en el capítulo 5 y de la estructura FOR, la fomra DO-CONTINUE, recientemente descrita, sugiere que la iteración empieza con VAR, tomando inicialmente el valor de INV, luego VAR se incrementa INCR con cada iteración, y la iteración se repite hasta que VAR pase a ENDV. Esta interpretación es básicamente correcta, pero es esencial que se conozca su significado exacto.

El diagrama de flujo para la implementación del lazo DO generalizado, aparece en la figura 12-7. Con la ejecución de la proposición DO se evalúan los valores de INV, ENDV e INCR, a VAR se le asigna el valor inicial INV y se evalúa el *contador de iteraciones*, que se designa por KOUNT. Normalmente, se calcula el contador de iteraciones KOUNT como el resultado a valor entero de (ENDV − INV+ INCR)/INCR. Por ejemplo,

```
DO 100 K = 2, 13, 3          deja   KOUNT =  4
DO 200 K = 2.0, 1.0, −0.05   deja   KOUNT = 21
```

Para satisfacer situaciones en las cuales INV > ENDV e INCR es positivo, o en donde INV < ENDV e INCR es negativo (es decir, donde VAR ha pasado inicialmente el valor ENDV) a KOUNT se le asigna el valor entero mayor que cero de (ENDV − INV + INCR)/INCR. Hay que hacer hincapié en que este número KOUNT da el número de veces que se va a ejecutar el lazo.

Como lo indica la figura 12-7, antes de ejecutar el lazo se prueba si KOUNT es cero. Si es diferente a cero, se ejecuta el lazo y VAR se incrementa en INCR y el contador de iteraciones KOUNT se disminuye en 1.



Implementación de un ciclo generalizado DO

Diagrama de flujo equivalente usando una caja DO

Fig. 12-7

Cuando KOUNT = 0, se transfiere el control a la primera proposición ejecutable que siga a la proposición CONTINUE.

EJEMPLO 12.6

(*a*) Suponga que se quieren imprimir los enteros 100, 99,..., 1 en este orden. Usando el lazo DO generalizado se tiene

```
      DO 500 K = 100, 1, -1
            WRITE(6, 30) K
   30       FORMAT(1X, I5)
  500 CONTINUE
```

(*b*) Usando el lazo DO generalizado por el problema del ejemplo 12.4, se tiene

```
      DO 999 X = -4.0, 4.0, 0.5
            Y = 2.0*X**2 - 3.0*X - 5.0
            WRITE(6, 10) X, Y
   10       FORMAT(2(3X, F10.3))
  999 CONTINUE
```

Las propiedades del lazo DO generalizado se pueden resumir así:

1. Como es usual, si se omite INCR, se supone 1.
2. El tipo de los valores de los parámetros del lazo, INV, ENDV e INCR, no necesitan ser del mismo tipo de VAR. En tal caso, los valores de los parámetros se convierten primero al tipo de VAR.

3. Como INCR puede ser negativo, es posible que VAR disminuya en valor. En cualquier caso, el lazo se ejecuta para cada valor de VAR, hasta que pase el valor final ENDV.
4. El valor de la variable del lazo, VAR, no se puede alterar dentro del cuerpo del lazo DO, excepto por su propio mecanismo de incremento.
5. La variable del lazo queda definida siempre y cuando se transfiera el control desde el lazo DO. Esto difiere con el índice del lazo DO básico (no estructurado), que queda definido solamente en caso de una salida anormal desde el lazo.]
6. Si INV > ENDV e INCR es positivo, o si INV < ENDV e INCR es negativo, KOUNT tendrá un valor inicial de cero y no se ejecutará el cuerpo del lazo.
7. No es permitido transferir el control desde fuera de un lazo DO generalizado a un punto intermedio de él (tal como sucede en el lazo DO básico indizado.) Las reglas para anidamiento de lazos DO discutidas en la sección 5.9, tienen también aplicación aquí.
8. Los lazos DO y las estructuras IF no se pueden traslapar. Sin embargo, un lazo DO puede estar contenido en un bloque de proposiciones de una estructura IF y viceversa.

Nota. Si la variable del lazo VAR es una variable entera, entonces la ejecución del lazo DO generalizado es exactamente como la de la estructura FOR, discutida anteriormente. En efecto, el lazo DO generalizado se compone tal como el lazo DO básico indizado, excepto que (*a*) INCR puede ser negativo; (*b*) los parámetros del lazo INV, ENDV e INCR pueden ser expresiones; y (*c*) la prueba es hecha antes de la ejecución del lazo.

Problemas resueltos

ESTRUCTURAS IF

12.1 Suponga que la TASA de interés sobre un préstamo AMT es 7% si AMT ≤ $10 000.00, pero es 6% si AMT excede a $10 000.00. Suponga que INT es una variable real y AMT está en memoria; escriba un segmento de programa que encuentre el interés INT sobre el préstamo.

La figura 12-8 contiene tal programa. (Compárelo con el programa no estructurado de la página 84.)

```
      IF(AMT.LE.10000.0) THEN
            RATE = 0.07
        ELSE
            RATE = 0.06
      ENDIF
      INT = AMT*RATE
      WRITE(6, 10) AMT, RATE, INT
   10 FORMAT(3(5X, F10.2))
```

Fig. 12-8

```
      IF(TYPE.EQ.1) THEN
            NET = PAY - 9.75
        ELSE IF(TYPE.EQ.2) THEN
            NET = PAY - 16.25
        ELSE IF(TYPE.EQ.3) THEN
            NET = PAY - 24.50
      ENDIF
      WRITE(6, 20) ID, NET
   20 FORMAT(1X, I5, 3X, F12.2)
```

Fig. 12-9

12.2 Suponga que la prima de seguro de salubridad se deduce del salario de un empleado, de acuerdo con el siguiente plan (donde TYPE es una variable entera):

$$\text{Prima} = \begin{cases} 9.75 & \text{si TYPE} = 1 \quad \text{(soltero)} \\ 16.25 & \text{si TYPE} = 2 \quad \text{(casado sin niños)} \\ 24.50 & \text{si TYPE} = 3 \quad \text{(casado con niños)} \end{cases}$$

Escriba un segmento de programa que deduzca la prima del salario de un empleado, suponiendo que PAY, TYPE e ID están en memoria (y NET es una variable real).

El diagrama de flujo para tal programa y una versión en Fortran no estructurado aparece en la página La figura 12-9 contiene una versión en Fortran estructurado, empleando la alternativa múltiple de la estructura IF.

12.3 Escriba un segmento de programa, que lea un entero M y lo imprima, ya sea positivo, negativo o cero.

La figura 12-10 contiene dicho programa, empleando la alternativa múltiple de la estructura IF.

```
      READ(5, 10) M
10    FORMAT(I5)
      WRITE(6, 20) M
20    FORMAT(1X, I5, 2X, 'IS')
      IF(M.GT.0) THEN
            WRITE(6, 30)
30          FORMAT(1X, 'POSITIVO')
       ELSE IF(M.EQ.0) THEN
            WRITE(6, 40)
40          FORMAT(1X, 'CERO')
       ELSE
            WRITE(6, 50)
50          FORMAT(1X, 'NEGATIVO')
      ENDIF
```

Fig. 12-10

```
      READ(5, 10) A, B, C
10    FORMAT(3F10.2)
      WRITE(6, 20) A, B, C
20    FORMAT(1X, 'SIDES', 3(2X, F10.2))
      IF(A.EQ.B.AND.B.EQ.C) THEN
            WRITE(6, 30)
30          FORMAT(1X, 'EQUILATERAL')
       ELSE IF(A.EQ.B.OR.A.EQ.C.OR.B.EQ.C) THEN
            WRITE(6, 40)
40          FORMAT(1X, 'ISOSCELES')
      ENDIF
```

Fig. 12-11

12.4 Escriba un segmento de programa que lea las longitudes A, B y C de un triángulo T, e imprima si T es equilátero o isósceles.

T es equilátero si A = B y B = C. T es isósceles si A = B ó A = C ó B = C. La figura 12-1 contiene dicho programa. (Se usan los relacionadores lógicos .AND. y .OR., discutidos en el capítulo 9.)

ESTRUCTURAS DO

12.5 Encuentre el contador de iteraciones KOUNT para cada proposición DO.

(*a*) DO 100 K = 4, 20, 3 (*c*) DO 300 K = 4, −9, −2
(*b*) DO 200 X = −4, 4, 0.6 (*d*) DO 400 X = 5, −2, 0.2

Primero, encuentre el cociente Q = (ENDV − INV + INCR)/INCR. Sea J el valor entero de Q. Entonces KOUNT es J o cero, dependiendo de cuál sea mayor.

(*a*) Q = (20 − 4 + 3)/3 = 19/3; así J = 6. De donde KOUNT = 6.
(*b*) Q = (4 + 4 + 0.6)/0.6 = 8.6/0.6; así J = 14. De Donde KOUNT = 14.
(*c*) Q = (−9 − 4 − 2)/−2 = −15/−2; así J = 7. De donde KOUNT = 7.
(*d*) Q = (−2 − 5 + 0.2)/0.2 = −6.8/0.2; así J = −34. De donde KOUNT = 0.

12.6 Encuentre el valor final de K y L.

(*a*)
```
      DO 100 K = 1, 5
         L = K**2
100   CONTINUE
      K = 2*K
```

(*b*)
```
      DO 200 K = 5, 1
         L = K**2
200   CONTINUE
      K = 2*K
```

(*a*) El lazo DO se ejecuta para K = 1, 2, 3, 4 y 5. De donde el valor final de L es $5^2 = 25$. Sin embargo, cuando el control se transfiere fuera del lazo, K se incrementa a 6. Entonces, 12 es el valor final de K.

(*b*) El lazo DO no se ejecuta, puesto que el contador de iteraciones KOUNT = 0; luego L no queda definido. Sin embargo, a K se le asigna el valor inicial de 5, cuando se ejecuta la proposición DO. Luego 10 es el valor final de K.

12.7 Suponga $z = x^3 - y^2$. Escriba un segmento de programa que encuentre z para valores de x e y variando desde -4 a 4 en pasos de 0.5.

La figura 12-12 contiene dicho programa, que emplea lazos DO generalizados anidados. [Compare este programa con su similar del problema 5.11(*b*).]

```
      DO 200 X = -4, 4, 0.5
             DO 100 Y = -4, 4, 0.5
                    Z = X**3 - Y**2
                    WRITE(6, 10) X, Y, Z
 10                 FORMAT(3(2X, F10.2))
100          CONTINUE
200   CONTINUE
```

Fig. 12-12

```
FOR(J = N, K, -1) DO
        A(J + 1) = A(J)
ENDFOR
A(K) = D
```

Fig. 12-13

12.8 Escriba un segmento de programa que emplee la estructura FOR para insertar un elemento DO en la K-ésima posición de un arreglo A(1), A(2),..., A(N).

Como se discutió en el problema 6.10, hay que mover el segmento de arreglo A(K),..., A(N) una posición antes de asignar D a A(K). Esto se consigue asignando primero A(N) a A(N + 1), entonces A(N − 1) a A(N) y así sucesivamente, hasta que A(K) se asigne a A(K + 1). La figura 12-13 contiene dicho programa.

12.9 Escriba un programa que emplee la estructura WHILE, para imprimir todos los enteros positivos impares menores que 100, omitiendo los múltiplos de 7:

1, 3, 5, 9, 11, 15, 17, 19, 23, . . . , 97, 99

La figura 12-14 contiene este programa. Observe que N se inicializa antes del lazo y se actualiza al final del mismo.

```
      N = 1
      WHILE(N.LT.100) DO
              K = (N*7)/7
              IF(K.EQ.N) GO TO 5
              WRITE(6, 10)
 10           FORMAT(10X, I2)
  5           N = N + 2
      ENDWHILE
```

Fig. 12-14

```
      DIMENSION SCORE(300)
      N = 1
      READ(5, 10) GRADE
 10   FORMAT(F8.0)
      WHILE(GRADE.GE.0.0) DO
              SCORE(N) = GRADE
              N = N + 1
              READ(5, 10)GRADE
      ENDWHILE
      N = N - 1
```

Fig. 12-15

12.10 Los puntajes de las pruebas para estudiantes se perforan en una tarjeta y el conjunto tiene una tarjeta indicadora perforada con un número negativo. Suponiendo que hay menos de 300 puntajes, escriba un segmento de programa que emplee la estructura WHILE para contar el número N de puntajes y almacenarlos en un arreglo SCORE.

La figura 12-15 muestra este problema. Obsérvese que N se disminuye en 1 después de que el control es transferido desde el lazo. (Por qué?)

Problemas suplementarios

ESTRUCTURAS IF Y DO

12.11 Suponga que J = 5 y K = 10. Encuentre el valor final de J y K después de ejecutar cada uno de los siguientes segmentos:

(*a*)
```
IF(J.LE.10) THEN
      K = K + 5
      J = J + K
ENDIF
J = J + K
J = 2*J
```

(*b*)
```
IF(J.LE.10) THEN
      K = K + 5
      J = J + K
ELSE
      J = J + K
ENDIF
J = 2*J
```

(*c*)
```
IF(J.GT.10) THEN
      K = K + 5
      J = J + K
ENDIF
J = J + K
J = 2*J
```

(*d*)
```
IF(J.GT.10) THEN
      K = K + 5
      J = J + K
ELSE
      J = J + K
ENDIF
J = 2*J
```

12.12 Considere los siguientes segmentos de programas:

(*i*)
```
M = 5
IF(J.LT.K) THEN
      J = J + 5
      M = M + 3
IF(J.LT.8) THEN
      M = M + 10
ELSE
      M = M + 20
ENDIF
```

(*ii*)
```
M = 5
IF(J.LT.K) THEN
      J = J + 5
      M = M + 3
ELSE IF(J.LT.8) THEN
      M = M + 10
ELSE
      M = M + 20
ENDIF
```

Encuentre el valor final de M para cada programa, si:

(*a*) J = 2, K = 5 (*b*) J = 2, K = 1 (*c*) J = 10, K = 15 (*d*) J = 10, K = 1

12.13 Encuentre el contador de iteraciones KOUNT para cada proposición DO.

(*a*) DO 100 K = −3, 19, 4
(*b*) DO 200 X = 4, −6, −0.3
(*c*) DO 300 K = 2, 25, −6
(*d*) DO 400 X = 1, −3, −0.2

12.14 Encuentre el valor final de J y M:

(*a*)
```
      DO 100 J = 2, 9, 3
            M = J + 5
100   CONTINUE
```

(*b*)
```
      DO 200 J = 9, 2, −5
            M = J + 5
200   CONTINUE
```

(*c*)
```
      DO 300 J = 9, 2, 3
            M = J + 5
300   CONTINUE
```

(*d*)
```
      DO 400 J = 2, 3, 9
            M = J + 5
400   CONTINUE
```

12.15 Encuentre el valor final de J, K y L:

(*a*)
```
      DO 100 J = 2, 6, 3
            DO 200 K = J, 4
                  L = J + K
200         CONTINUE
100   CONTINUE
```

(*b*)
```
J = 3
WHILE(J.LT.6) DO
      K = 2*J
      L = 3*J
      J = J + 2
ENDWHILE
```

PROGRAMAS

12.16 Al puntaje T de una prueba se le asigna la letra A, B, C, D o F, de acuerdo con:

$$T \geq 90, \quad 80 \leq T < 90, \quad 70 \leq T < 80, \quad 60 \leq T < 70, \quad \text{ó} \quad T < 60$$

Escriba un programa que lea un puntaje de prueba T y escriba su correspondiente letra.

12.17 Cada tarjeta de un paquete contiene el puntaje de prueba de un estudiante y el conjunto tiene una tarjeta indicadora con un número negativo. Escriba un programa que encuentre el número N de estudiantes que tomaron el examen, el número M de estudiantes que tienen un puntaje perfecto (es decir, 100) y el número L de estudiantes que perdieron la prueba (es decir < 60).

12.18 Escriba un programa que calcule las raíces reales de la ecuación cuadrática

$$ax^2 + bx + c = 0$$

dados los coeficientes A, B, C con $A \neq 0$. (Compárelo con el programa de la página 90.)

12.19 Escriba un programa que lea tres números positivos A, B, C y determine si A, B y C pueden ser las longitudes de los lados de un triángulo. Si lo son, calcule el perímetro del triángulo; si no, escriba el mensaje "no es un triángulo". (Sugerencia: A, B y C no forman un triángulo, si y solamente sí, un lado es mayor o igual que la suma de los otros dos.) Pruebe el programa con los datos:

12.20 Escriba un segmento de programa que emplee un lazo DO generalizado para insertar un elemento D en la K-ésima posición de un arreglo A(1), A(2),..., A(n). (Compare con los problemas 12.8 y 6.10.)

12.21 Encuentre el número de puntos con coordinadas enteras que se encuentren dentro de:
(a) el círculo $x^2 + y^2 = 50$ (b) la eclipse $2x^2 + 3y^2 = 100$
(Compare con el problema 5.12.)

12.22 Escriba un programa que imprima el número 20 veinte veces, el número 19 diecinueve veces, el número 18 dieciocho veces y así, sucesivamente.

12.23 Cada tarjeta de un paquete contiene una tarjeta indicadora en blanco. Escriba un programa que encuentre el número de enteros pares e impares.

12.24 Considere la ecuación $y = x^2 - 4x + 6$. Escriba un programa que encuentre y para valores de x desde I hasta J con incremento D, donde I, J y D están en una tarjeta e $I < J$ y $D > 0$.

12.25 Suponga que se depositan \$2 000.00 en una cuenta de ahorros con un interés compuesto anual del 7%. Escriba un programa que emplee la estructura WHILE para determinar el número de años necesarios para que la cuenta llegue a tener \$5 000.00.

12.26 Un almacén decide hacer un descuento sobre el precio PRICE de un conjunto de televisores a color de acuerdo con el tamaño SIZE en pulgadas del TV como sigue:

$$\text{Descuento} = \begin{cases} \$\ 40.00 \text{ si SIZE} = 12 \text{ ó } 15 \\ \$\ 70.00 \text{ si SIZE} = 17 \text{ ó } 19 \\ \$110.00 \text{ si SIZE} = 21 \text{ ó } 23 \\ \$140.00 \text{ si SIZE} = 25 \end{cases}$$

Hay un descuento adicional del 3% sobre el precio PRICE, si 300 ≤ PRICE < 400 y del 4% si PRICE > 400. Escriba un programa que lea el número de identificación ID de un TV y su tamaño SIZE y precio PRICE. Además, el costo neto NET del TV. (Suponga que SIZE es una variable entera y NET es una variable real.)

12.27 Hay cuatro candidatos para un puesto en la junta de educación. Una persona vota perforando 1, 2, 3 ó 4 en una tarjeta. El paquete tendrá una guia sobre la que se perfora −1. Escriba un programa que cuente el número N de personas que votaron y el número de votos por cada candidato.

12.28 Repita el problema 12.27, excepto que ahora el programa debe también encontrar el ganador de la elección.

12.29 Repita el problema 12.27, excepto que ahora hay cinco candidatos para dos posiciones, debiendo el programa encontrar también los dos ganadores.

Respuestas a problemas seleccionados

12.11 (*a*) 70, 15 (*b*) 40, 15 (*c*) 30, 10 (*d*) 30, 10

12.12 (*a*) (*i*) 18 (*ii*) 8 (*c*) (*i*) 28 (*ii*) 8
(*b*) (*i*) 15 (*ii*) 15 (*d*) (*i*) 25 (*ii*) 25

12.13 (*a*) 5 (*b*) 33 (*c*) 0 (*d*) 20

12.14 (*a*) J = 11, M = 13 (*c*) J = 9, M no definido
(*b*) J = −1, M = 9 (*d*) J = 11, M = 7

12.15 (*a*) J = 8, K = 5, L = 4 (*b*) J = 7, K = 10, L = 15

Apéndice A

Funciones de biblioteca

Este apéndice da una lista de funciones de biblioteca (internas) que están disponibles para la mayoría de los sistemas Fortran. Los argumentos de las funciones en la tabla A-1 están denotados por los siguientes símbolos, en los cuales también se especifica su tipo:

J, K	tipo entero	DX, DY	tipo doble precisión
X, Y	tipo real	C	tipo complejo

La palabra *doble* se usará por doble precisión. El signo de un argumento a, denotado por sgn(a), está definido como:

$$\text{sgn}(a) = \begin{cases} 1 & \text{si } a \geq 0 \\ -1 & \text{si } a < 0 \end{cases}$$

La mayor función entera, denotada por $[a]$, da el entero mayor $\leq a$.

Tabla A-1

Descripción	Definición	Nombre de la función con los argumentos	Tipo de la función
Raíz cuadrada	$\sqrt{a}$	SQRT(X) DSQRT(DX) CSQRT(C)	Real Doble Complejo
Exponencial	e^a	EXP(X) DEXP(DX) CEXP(C)	Real Doble Complejo
Logaritmo natural	$\log_e(a)$	ALOG(X) DLOG(DX) CLOG(C)	Real Doble Complejo
Logaritmo decimal	$\log_{10}(a)$	ALOG10(X) DLOG10(DX)	Real Doble
Valor absoluto	$\lvert a \rvert$	IABS(J) ABS(X) DABS(DX)	Entero Real Doble
Funciones de un argumento complejo $C = x + iy$: Magnitud Complejo conjugado Parte real Parte imaginaria	 $\sqrt{x^2 + y^2}$ $x - iy$ x y	 CABS(C) CONJG(C) REAL(C) AIMAG(C)	 Real Complejo Real Real

Tabla A-1 (*Continuación*)

Descripción	Definición	Nombre de la función con los argumentos	Tipo de la función
Forma compleja de dos valores reales x, y	$x + iy$	CMPLX(X, Y)	Complejo
Conversión de tipo	Conversión de entero a real Conversión de entero a doble precisión Conversión de real a doble precisión Conversión de doble precisión a real por truncamiento Conversion de real a entero por truncamiento	FLOAT(J) DFLOAT(J) DBLE(X) SNGL(DX) IFIX(X)	Real Doble Doble Real Entero
Máximo valor	$\text{Max}(a_1, a_2, \ldots)$	MAX0(J, K, ...) MAX1(X, Y, ...) AMAX0(J, K, ...) AMAX1(X, Y, ...) DMAX1(DX, DY, ...)	Entero Entero Real Real Doble
Mínimo valor	$\text{Min}(a_1, a_2, \ldots)$	MIN0(J, K, ...) MIN1(X, Y, ...) AMIN0(J, K, ...) AMIN1(X, Y, ...) DMIN1(DX, DY, ...)	Entero Entero Real Real Doble
Diferencia positiva o cero de (a_1, a_2)	$a_1 - \min(a_1, a_2)$	IDIM(J, K) DIM(X, Y)	Entero Real
Funciones trigonométricas con argumento en en radianes	$\text{sen}(a)$	SIN(X) DSIN(DX) CSIN(C)	Real Doble Complejo
	$\cos(a)$	COS(X) DCOS(DX) CCOS(C)	Real Doble Complejo
	$\tan(a)$	TAN(X) DTAN(DX)	Real Doble
Transferencia de signo de (a_1, a_2)	$\lvert a_1 \rvert \cdot \text{sgn}(a_2)$	ISIGN(J, K) SIGN(X, Y) DSIGN(DX, DY)	Entero Real Doble
Módulo de truncamiento	$\text{sgn}(a) \cdot [\lvert a \rvert]$	INT(X) AINT(X) IDINT(DX)	Entero Real Entero
Determinación del residuo entre a_1 y a_2, a partir del mínimo valor absoluto	$a_1 - \text{sgn}(b)[\lvert b \rvert] \cdot a_2$, donde $b = a_1/a_2$	MOD(J, K) AMOD(X, Y) DMOD(DX, DY)	Entero Real Doble

Apéndice B

Representación interna de datos

INTRODUCCION

Cada celda de memoria está compuesta de aparatos que son biestables por naturaleza, es decir, aparatos que pueden estar en dos estados. Puesto que esos estados se pueden representar por los dígitos binarios 0 y 1, cada aparato biestable es capaz de almacenar un *bit* (por *dí*gito *bi*nario) de información. Generalmente se usa una colección o secuencia de bits para representar una unidad de información.

Puesto que una secuencia de dígitos binarios se puede considerar como un número binario, la representación de números en el sistema binario es muy importante en la ciencia de los computadores. La tabla B-1 lista la representación binaria de los primeros dieciseis enteros positivos. Algunas veces, se usa el subíndice 2 para distinguir un número binario de un número decimal; por ejemplo,

$(101101)_2$ significa el número binario 101101

el cual tiene la representación decimal de 45. Una discusión general de la conversión de números decimales a números binarios y viceversa está más allá del marco de este apéndice.

Tabla B-1

Número decimal	Número binario
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

PALABRAS Y BYTES

La unidad básica de información que es almacenada o transmitida desde una celda de memoria, se llama *palabra*. Algunos tamaños populares de palabra son 24, 30, 32, 36, 48 y 60 bits. Hay computadores con

palabras de longitud fija y computadores con palabras de longitud variable. Sin embargo, cada palabra en la memoria debe tener su propia dirección. Más aún, cuando entramos/sacamos una palabra desde la memoria, todos sus bits se transmiten simultáneamente.

Algunos computadores también permiten que parte de una palabra se almacene o se transmita desde la memoria. Esta subunidad de la palabra se llama un *byte*. Generalmente un byte consiste de 6 u 8 bits y se pueden direccionar individualmente. Las máquinas con tales características se dice que son *direccionables por bytes*. La serie CDC 6000-7000 tiene palabras compuestas de 10 bytes de 6 bits lo que da una palabra de 60 bits. Para propósitos ilustrativos, supondremos que nuestra máquina es semejante a la serie IBM 360/370, la cual tiene palabras compuestas de 4 bytes de 8 bits, dando una palabra de 32 bits como sigue:

Palabra

Byte 0	Byte 1	Byte 2	Byte 3

Inclusive en las máquinas direccionables por byte, es aún la palabra una importante unidad de información, siendo su dirección, la dirección del byte que está más a la izquierda de la palabra.

DATOS DE CARACTERES

Normalmente, los caracteres reconocidos por cualquier compilador Fortran, incluyen los 48 caracteres mencionados en el capítulo 2; es decir, los 10 dígitos, 26 caracteres alfabéticos y 12 caracteres especiales. Puesto que

$$2^5 < 48 < 2^6$$

se requiere un mínimo de 6 bits para representar esos caracteres. Cualquier código de 6 bits puede representar hasta 64 caracteres distintos permitiendo un máximo de $64-36=28$ caracteres especiales. La mayoría de los computadores de la tercera generación utiliza códigos de 8 bits. Esto permitirá hasta 256 caracteres, lo cual es totalmente adecuado.

Los caracteres alfabéticos están generalmente codificados en orden alfabético, es decir, el código de A es más pequeño que el código de B y así sucesivamente. Esta codificación forma una secuencia que puede ser utilizada para ordenar datos alfabéticos. (Ver capítulo 9.)

El código de 8 bits (6 bits) que corresponde a un caracter dado se conforma generalmente por una porción zonificada, generalmente los 4 (3) bits que están más hacia la izquierda, y una porción numérica, generalmente los 4 (3) bits que están más hacia la derecha. (Ver figura B-1.) El código de caracteres utilizado por las máquinas IBM 360/370 se llama EBCDIC (código de intercambio a decimal codificado en binario extendido.) La tabla B-2 da el código EBCDIC para los 26 caracteres alfabéticos y los 10 dígitos. Obsérvese

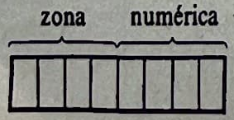


Fig. B-1

que el código para cada dígito contiene 1111 en la porción zonificada y su representación binaria en la porción numérica, puesto que no más de 4 caracteres de 8 bits se pueden almacenar en una palabra de 32 bits, esto explica la razón por la que la capacidad de caracteres de una máquina IBM 360/370 es de $M=4$.

Tabla B-2 Códigos EBCDIC

Caract.	EBCDIC	Caract.	EBCDIC	Caract.	EBCDIC	Caract.	EBCDIC
	zona numérica		zona numérica		zona numérica		zona numérica
A	1100 0001	J	1101 0001			0	1111 0000
B	0010	K	0010			1	0001
C	0011	L	0011	S	1110 0010	2	0010
D	0100	M	0100	T	0011	3	0011
E	0101	N	0101	U	0100	4	0100
F	0110	O	0110	V	0101	5	0101
G	0111	P	0111	W	0110	6	0110
H	1000	Q	1000	X	0111	7	0111
I	1100 1001	R	1101 1001	Y	1000	8	1000
				Z	1110 1001	9	1111 1001

DATOS NUMERICOS

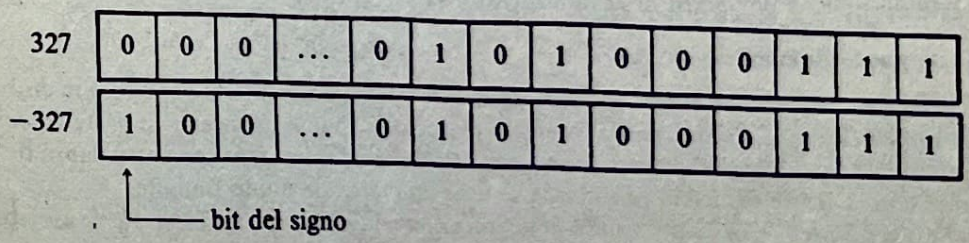
Hay esencialmente dos formas diferentes para codificar datos numéricos; codificar la cantidad o codificar el dígito individual. Discutiremos primero la codificación de la cantidad.

Codificación de la cantidad

Para los cálculos, Fortran provee aritmética entera y aritmética de punto flotante. La representación interna de esos dos tipos de datos numéricos es diferente.

(*a*) Enteros

Una forma directa de representar un entero en una celda de memoria es utilizar su representación binaria. Asumiendo celdas de memoria con 32 bits, con el primer bit reservado para el signo, los enteros 327 y -327 serán almacenados como sigue:



En consecuencia, el mayor entero que se puede almacenar en una palabra de 32 bits es

$$0\underbrace{(111\ldots1111)}_{\text{31 unos}}{}_2$$

que representa $2^{31}-1$. Cualquier intento de almacenar enteros mayores que esta capacidad causará sobreflujo y probablemente se perderán los dígitos más significativos.

Por razones computacionales, los números negativos se almacenan algunas veces en memoria, utilizando la representación denominada complemento *a uno* o complemento *radical*. Discutiremos primero esas representaciones en el sistema decimal donde son llamados complemento *a nueve* y complemento *a diez*, respectivamente. El complemento a nueve de un número decimal se obtiene restando cada dígito

decimal de nueve, y el complemento a diez de un número es su complemento a nueve más a uno. Por ejemplo,

Número decimal:	327	123	320
Complemento a nueve:	672	876	679
Complemento a diez:	673	877	680

Los complementos correspondientes para números binarios son llamados el complemento *a uno* y complemento *a dos*, respectivamente. Análogamente, el complemento a uno de un número binario se obtiene restando cada dígito binario de uno (es decir, convirtiendo cada cero en uno y cada uno en cero), y el complemento a dos de un número binario es su complemento a uno más uno. Por ejemplo,

Número binario:	11001	10111	10100
Complemento a uno:	00110	01000	01011
Complemento a dos:	00111	01001	01100

El representar número negativo mediante complemento nos permite utilizar la resta por medio de la suma (puesto que la multiplicación es una suma repetida y la división una resta repetida, toda la aritmética entera se puede realizar mediante la suma.) Indicaremos como trabaja esta aritmética con un ejemplo:

Aritmética ordinaria	Sistema decimal utilizando complemento a nueve	Sistema binario utilizando complemento a uno con una palabra de 32 bits
705	705	00000 . . . 0001011000001
−327	+672	+11111 . . . 1111010111000
378	(1)377	(1)00000 . . . 0000101111001
	→1	→1
	378	00000 . . . 0000101111010

Observe que se borra el uno que está más a la izquierda y luego se añade a la suma, como se indica mediante las flechas. Esto se denomina "acarreo de fin de ciclo". Los detalles de esta aritmética están más allá del propósito de éste apéndice y por tanto no serán discutidos.

(*b*) Números de punto flotante

La suma, resta y multiplicación entera conducirá siempre a resultados enteros exactos. Sin embargo, la división entera también conduce a un resultado entero y por tanto no se almacena su parte fraccionaria. Además, ningún número con valor absoluto que exceda a 2^{31} se puede almacenar en una palabra de 32 bits. Esas limitaciones son la razón principal por la que se usan números de punto flotante.

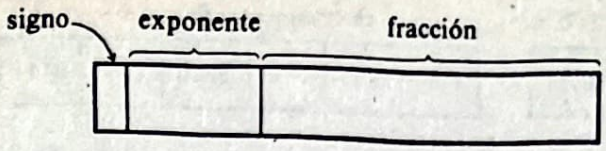
Consideremos el número −32.75 utilizando notación exponencial normalizada, se puede escribir como

$$-0.3275 \times 10^2 \quad \text{con} \begin{cases} \text{signo} - \\ \text{fracción } 0.3275 \\ \text{exponente } 2 \text{ (base } 10) \end{cases}$$

Se puede utilizar una representación similar para números binarios. Por ejemplo, el número decimal − 32.75 se puede escribir como el número binario $(-100000.11)_2$ y es normalizado como

$$-0.10000011 \times 2^6 \quad \text{con} \begin{cases} \text{signo} - \\ \text{fracción } 0.10000011 \\ \text{exponente } 6 \text{ (base } 2) \end{cases}$$

Claramente cada número real se puede normalizar para que tenga un signo, una fracción binaria y un exponente binario único. Por tanto, se puede almacenar internamente como sigue:



En nuestra palabra de 32 bits, suponemos un bit para el signo, 7 bits para el exponente y $32-8=24$ bits para la fracción binaria. Hacemos notar que cualquier número real que se almacene utilizando campo E o campo F tendrá la misma representación interna anterior.

Puesto que un exponente puede ser negativo se deben hacer algunos arreglos para tener en cuenta su signo en el campo de 7 bits. En lugar de utilizar un bit para denotar el signo del exponente, convertimos el exponente verdadero a la así llamada *característica*, añadiendo $(100000)_2 = 2^6 = 64$ y almacenando la característica en lugar del exponente verdadero. La siguiente tabla muestra la relación entre el exponente verdadero y su característica.

Exponente verdadero:	−64	−63	−62	−61	...	−1	0	1	...	63
Característica:	0	1	2	3	...	63	64	65	...	127

(En el caso de que usáramos para el exponente un campo de 8 bits, sumariamos $2^7 = 128$ al exponente verdadero para obtener su característica y así sucesivamente.)

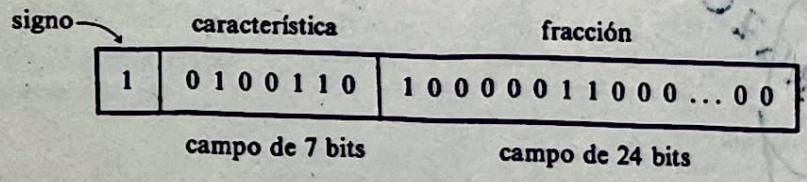
Consideremos de nuevo el número -32.75, cuya representación binaria normalizada es

$$-0.10000011 \times 2^6$$

Aquí el exponente verdadero es 6 y así su característica es

$$6 + 64 = 70 = (100110)_2$$

Luego, -32.75 se almacena internamente como



Se puede ver que el mayor número de punto flotante que se puede almacenar en una palabra de 32 bits (utilizando forma binaria flotante) es 2^{63} y el número más pequeño es 2^{-64}. Cualquier intento de almacenar números mayores que 2^{63}, o menor que 2^{-64} causará sobreflujo (o saturación). A causa de la longitud finita de la palabra, los números de punto flotante pueden tener representaciones internas inexactas.

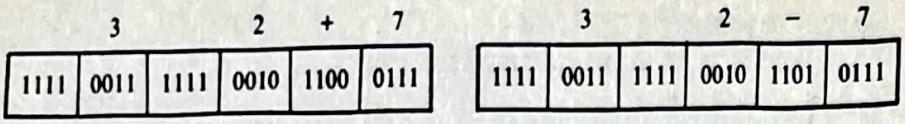
Finalmente, recalcamos que en la forma de punto flotante, no se usa la notación de complemento para fracciones negativas.

Codificación de digitos individuales

Otra forma de codificar datos numéricos es utilizar código de caracteres con formato de zona para cada uno de sus dígitos. En tal caso el signo es colocado en el campo de zona del dígito que esté más hacia la derecha como sigue:

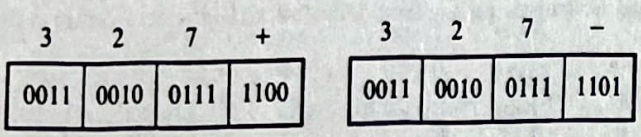
zona	dígito	zona	dígito	...	zona	dígito	signo	dígito

En EBCDIC, los signos se representan mediante 1100 y 1101, respectivamente. Utilizando la tabla B-2 para los códigos EBCDIC de los dígitos, los números 327 y -327 son codificados como sigue

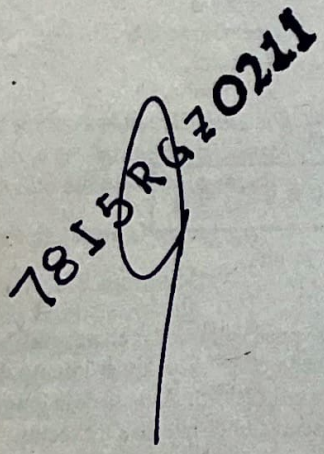


La anterior es la forma como los datos numéricos se representan en una entrada y salida. Sin embargo, cuando se usan los datos numéricos para cálculos, ellos se convierten a una forma diferente para facilidad de procesamiento. Una de las formas es el sistema decimal codificado en binario (BCD).

La codificación BCD de los números es similar a la codificación anterior, excepto que no se usan los campos de zona y el código del signo aparece en el campo que está más hacia la derecha. En otras palabras, se usa una representación binaria de 4 bits para cada dígito decimal del número con el código del signo al final del número. En consecuencia, en BCD, 327 y −327 son codificados como



La suma en BCD utiliza suma binaria para los dígitos decimales individuales. Sin embargo, cuando una suma excede a 9, se acarrea una unidad al siguiente lugar como en aritmética decimal ordinaria. Cualquier otra discusión acerca de esta aritmética escapa del marco de este apéndice.



Indice

ABS, 26
Alfanumérico, 19
Algoritmo, 77, 91
Almacenamiento de números, 3-4
AND, 225
Aritmética:
  expresión, 23
  IF, 84
  operaciones, 22
  proposiciones, 19
  proposición de asignación, 299
  proposición que define FUNCION, 179, 180
  unidad, 12
Aritmética de punto flotante, 22
Aritméticas del computador, 30
Argumento, 175-176-177-178-180-182-183-185-187
Arreglo, 140
  bidimensional, 147
  en subprogramas, 178, 183-184
  entrada/salida, 151
  expresiones para subíndice, 143
  lineal, 140
  multidimensional, 147
  nombre, 141
  orden de almacenamiento, 148
Arreglo tridimensional, 148

BCD, 310
Bit, 305
Búsqueda:
  binaria, 198
  lineal, 162, 198
  secuencial, 162, 198
Blanco, 19
BLOCK DATA, 281
Bloque ELSE, 290
Bloque IF, 289
Byte, 305-306

Cadena, 217
Campo, 43, 49, 195
  campo A, 219
  campo D, 266
  campo E, 46, 53
  campo F, 45, 52
  campo G, 242
  campo H, 223
  campo I, 44, 51
  campo L, 54
  campo T, 241
  campo X, 48, 51
  especificación, 43
  literal, 54
  tamaño, 43
Campo hollerith; 223
Carácter:
  alfanumérico, 19
  cadena, 217
  capacidad, 217
  conjunto, 19
  dato, 217, 306
Característica, 309
Ciclo, 85
Ciclo DO, 114
  anidado, 126
  implícito, 153
  generalizado, 295
  salida de, 121
Ciclo infinito, 88
Ciclo WHILE, 292
Clasificación:
  burbuja, 195
  intercalación, 197
  intercambio, 196
  interna, 195
  selección, 196
Clave, 197
Código de formato, 43
Columna, 148
Columna de continuación, 1
Compilador, 13
COMPLEX, 269
Complemento a uno, 308
Complemento de una expresión lógica, 289
Computador de programa almacenado, 12
Constante:
  compleja, 269
  doble precisión, 266
  entera, 19
  lógica, 224
  punto fijo, 19
  punto flotante, 20
  real, 20
Contador de iteraciones, 295

Control de carro, 48-49
Cuerpo del ciclo DO, 114, 118

DATA:
  BLOCK, 281
  proposición, 217, 239
Dato:
  FORTRAN, 3
  paquete, 3
  representación interna, 305
Decimal codificado en binario, 310
Decisión, 8
Depuración, 144
Diagrama de Flujo, 77
Dígito binario, 305
Dimensión variable, 178
Dirección, 12
Distribución, frecuencia, 164
DBLE, 267
DFLOAT, 267
Doble precisión, 265
  aritmética, 266
  constante, 265-266
  entrada/salida, 266
  función de biblioteca, 267
  subprogramas, 267
  variable, 265
DO anidados, 126, 151
DO implícitos anidados, 154

EBCDIC, 307
Eco prueba, 144
Ecuación lineal, 209
Eficiencia, 194
ELSE, 289
ELSE ablocado, 290
ELSE IF, 291
ENDFOR, 294
ENDIF, 288
ENDWHILE, 293
Ensamblador, 13
Entera:
  aritmética, 22
  constante, 19
  división, 22
  variable, 21
Entrada/salida:
  con formato, 42, 44, 49
  sin formato, 41
Entrada múltiple, 272
Entrada/salida con formato libre, 41
Error:
  conversión, 30
  redondeo, 30
Errores, 30
Escalar, 140, 205
Especificación:
  de campo, 43
  de E/S, 43
Especificaciones de campos de entrada:
  campo E, 46
  campo F, 45
  campo I, 44
  campo X, 48
Especificaciones de campos de salida:
  campo E, 53
  campo F, 52
  campo I, 51
  campo X, 51
Estructuras de control de ciclo, 292
Estructura FOR, 293
Estructuras IF, 288
  alternativa doble, 289
  alternativa múltiple, 291
  alternativa sencilla, 288-289
Estructura WHILE, 292
Expresión:
  aritmética, 23
  lógica, 224
  de relación, 82
Expresión libre de paréntesis, 23

Factor de escala, 244
Factor de repetición, 239
Factor P, 244
FLOAT, 27
Forma decimal, 20
Forma E, 20
Forma exponencial, 20
  normalizada, 53
Formato a tiempo de ejecución, 246
FORTRAN estructurado, 288
Función CMPLX, 269
Función del mayor entero, 305
Funciones de biblioteca, 26, 303
Funciones matemáticas internas, 26
FUNCION:
  proposición de definición, 174
  subprograma, 172-173

GO TO asignado, 271
Gráfico, 248

Hoja de codificación, 1

IFIX, 26
Incremento, 118
Integración, 204
Intercalación, 198
Inventario, 170
IF. . . THEN. . ., 81, 288
IF lógico, 83
IF. . . THEN. . . ELSE. . ., 290
  incondicional, 78
  dentro de un ciclo DO, 183
Intervalo de bisección, 202

Jerarquía:
  de lenguajes de programación, 12
  de operaciones, 23

Justificado a la derecha, 43, 49
Justificado a la izquierda, 43

Lectura destructiva, 21
Lectura no destructiva, 21
Lenguaje compilador, 13
Lenguaje de alto nivel, 12-13
Lenguaje de computador, 11
Lenguaje de máquina, 12
Lenguaje ensamblador, 13
Lógica:
  constante, 224
  operador, 225
  proposición de asignación, 226
  variable, 224
LOGICAL, 224
Localización de almacenamiento, 12

Llamado de programa, 175

Mantisa, 20, 309
Matrices, 205
Matriz aumentada, 211
Media, 159
Memoria:
  elemento, 21
  unidad, 12
Método de Gauss-Jordan, 215-216
Método de Horner, 200
Método de Newton-Raphson, 214
Método Gaussiano, 210
Método de Sieve, 171
Mezcla de modo, 25
Modo de la operación, 23
Múltiple:
  entrada, 272
  registros, 55
  retorno, 272

Nombre de variable, 12, 21
NOT, 225
Numérica:
  constante, 19
  dato, 309
  integración, 204
Número complejo, 268
Número de aparato, 42
Número primo, 87

Operadores:
  aritmético, 22
  de relación, 82
  lógico, 225
Operadores de relación, 82
OR, 225

Palabra, 305-306
Paquete:
  FORTRAN, 3
Parámetros, 175-176
  ciclo, 118, 295
Perforación, 1
Programa fuente, 13
Programa objeto, 13
Proposición, 4, 29
Proposición:
  aritmética, 19, 29
  asignación, 29
  ejecutable, 13
  lógica, 226
  número, 3
  no ejecutable, 13
Proposición CALL, 183
Proposición COMMON, 273
  rotulado, 276
  no rotulado (blanco), 273
Proposición CONTINUE, 113-114
  necesidad de, 123-124
Proposición DIMENSION, 142
Proposición DO, 114, 118
  generalizado, 295
Proposición de declaración de tipo:
  COMPLEX, 269
  DOUBLE PRECISION, 265
  INTERGER, 21
  LOGICAL, 224
  REAL, 21
Proposición ejecutable ficticia, 113
Proposición END, 5, 13
Proposición EQUIVALENCE, 278
Proposición FORMAT, 44
Proposición FUNCION, 179
Proposición GO TO, 8, 78
  computado, 90
Proposición IF:
  aritmética, 84
  lógica, 83
Proposición IF ablocada, 288
Proposición IMPLICIT, 264
Proposición PRINT, 5, 41
Proposición READ, 41, 44
Proposición RETURN, 174, 273
Proposición WRITE, 48-49

Radicales, 309
Radicales menos uno, 309
Raíces de una ecuación, 203
Real:
  aritmética, 22-23
  constante, 20
  representación interna de, 307
  variable, 21
REAL, 21
Registro, 55, 195
Representación interna, 306
  datos alfanuméricos, 306
  datos numéricos, 307

Salida:
  aparato, 42
  con formato, 42

cadena, 49
unidad, 12
Salida anormal, 121
Salida normal, 121
Secuencia de Fibonacci, 160, 169
SGN, 303
Símbolo de la sumatoria, 207
Sistema triangular de ecuaciones, 209
Solución de ecuaciones, 201
Solución de una ecuación, 201
Subíndice, 140, 148
Subprograma:
complejo, 271
doble precisión, 267
FUNCTION, 172
SUBROUTINE, 181-182
Sustitución hacia atrás, 209

Tabla de jerarquía, 23
Tamaño de campo, 43
Tarjeta, 1
Tarjeta centinela, 97
Tarjeta de comentario, 3
Tarjeta de control, 3
Tarjeta de delimitación, 95
Tarjeta de encabezamiento, 95
Tarjeta de final de archivo, 3
Tarjeta ID, 3
Técnica de Programación, 194
Tipos de aritmética:
compleja, 270
doble precisión, 266
entera, 22
precisión sencilla, 266
real, 22
Transferencia de control:
condicional, 80
IF. . . THEN. . . ELSE, 290
incondicional, 78
dentro de un ciclo DO, 123

UCP, 12
Unidad central de proceso, 12
Unidad de control, 12
Unidad de entrada, 12

Valor de control, 118
Valor inicial, 118
Valor final, 118
Valor límite, 118
Variable, 12
Variable ficticia, 176
Variable índice, 114
Variable subindicada, 140
Variables de control del lazo, 292

LIBROS DE LA SERIE SCHAUM PUBLICADOS EN ESPAÑOL

MATEMATICAS/ESTADISTICA
Algebra elemental • Algebra elemental moderna • Algebra lineal • Algebra superior • Análisis de Fourier
Análisis vectorial • Cálculo diferencial e integral • Cálculo superior • Ecuaciones diferenciales
Ecuaciones diferenciales modernas • Espacio de estado y sistema lineal • Estadística
Estadística aplicada a la administración • Geometría analítica • Geometría descriptiva
Geometría diferencial • Geometría plana • Geometría proyectiva • Manual de fórmulas matemáticas
Matemáticas aplicadas a la ciencia y la tecnología • Matemáticas financieras • Matemáticas finitas
Matemáticas superiores para ingenieros y científicos • Matrices
Métodos cuantitativos para administración de empresas • Probabilidad y estadística con cálculo
Teoría de conjuntos y temas afines • Teoría de grupos • Topología • Transformadas de Laplace
Trigonometría • Variables complejas • Variables reales • Vibraciones mecánicas.

ECONOMIA/ADMINISTRACION DE EMPRESAS
Contabilidad I • Contabilidad II • Contabilidad intermedia I • Contabilidad intermedia II (en prensa)
Contabilidad de costos (en prensa) • Desarrollo económico • Economía internacional
Macroeconomía • Microeconomía • Publicidad

INGENIERIA
Acústica • Análisis estructural • Análisis estructural avanzado • Circuitos eléctricos
Circuitos electrónicos • Dinámica de los fluidos • Dinámica de Lagrange • Diseño de concreto armado
Diseño de máquinas • Ecuaciones básicas de las ciencias de la ingeniería • Líneas de transmisión
Materiales de construcción • Mecánica de los fluidos e hidráulica • Mecánica del medio continuo
Mecánica para ingenieros (Estática y Dinámica), 2a ed. • Mecánica teórica • Resistencia de materiales
Retroalimentación y sistemas de control • Termodinámica • Transferencia de calor

CIENCIAS NATURALES
Ciencias físicas • Física aplicada • Física general • Física moderna (en prensa) • Fisicoquímica
Genética • Óptica • Química general • Química orgánica.

COMPUTACION
Análisis numérico • Ciencias de las computadoras • Programación BASIC
Programación con FORTRAN

CIENCIAS SOCIALES
Introducción a la psicología

EDUCACION MEDIA - SECUNDARIA - PREPARATORIA - VOCACIONAL - PRE UNIVERSITARIA - C.O.U.
Fundamentos de física, I y II • Fundamentos de matemática, I y II • Fundamentos de química general
Fundamentos de química orgánica

SCHAUM... TU COMPAÑERO DE ESTUDIOS

84-8278-002-8